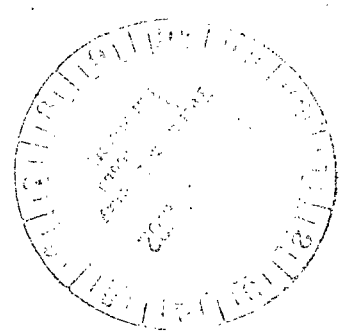
NASA Technical Paper 2086

September 1982

Nonlinear Optimization With Linear Constraints Using a Projection Method

Thomas Fox



NASA

NASA
Technical
Paper
2086

1982

Nonlinear Optimization With Linear Constraints Using a Projection Method

Thomas Fox
George C. Marshall Space Flight Center
Marshall Space Flight Center, Alabama

NASA
National Aeronautics
and Space Administration

**Scientific and Technical
Information Branch**

ACKNOWLEDGMENT

I would like to thank Dr. M. Lowe for his patience and the long hours of review, corrections, and suggestions that went into making this report possible. I would also like to thank Dr. L. Foster for his time and effort and Dr. John Glaese without whose suggestions I would probably be writing this report on an entirely different subject. To these gentlemen, I thank you.

TABLE OF CONTENTS

		Page
I.	INTRODUCTION	1
II.	STATEMENT OF THE PROBLEM	2
III.	FINDING A FEASIBLE POINT	4
IV.	UNCONSTRAINED OPTIMIZATION	13
V.	CONSTRAINED OPTIMIZATION	19
VI.	CONCLUSIONS	28
REFERENCES		29
APPENDIX A.	USERS GUIDE	31
APPENDIX B.	TEST PROBLEMS	35
APPENDIX C.	PROGRAM DESCRIPTION	41
APPENDIX D.	PROGRAM LISTING	45

LIST OF ILLUSTRATIONS

Figure	Title	Page
1.	Operations to update the projection matrix	3
2.	Nonlinear projection optimization	5
3.	Search vector and gradient at x^{i+1}	14
4.	Projection of $\nabla f(x)$ onto hyperplane g_i	22
5.	Graphical display of Farkas Theorem	24
6.	Intersection of search vector and constraints	26
B-1.	Optimal points for problems T5, T6, and T7	38
B-2.	Optimal point for problem T8	38
B-3.	Advantages of a nonlinear over a linear search	40
C-1.	Extremum estimation along search vector	42

TECHNICAL PAPER

NONLINEAR OPTIMIZATION WITH LINEAR CONSTRAINTS USING A PROJECTION METHOD

I. INTRODUCTION

The solution to the extremization problem with a nonlinear objective and linear equality and inequality constraints has application to many fields of science and business. Extremization means either maximization or minimization of an objective function with respect to a set of decision variables that may be required to satisfy a set of equality and inequality constraints. Finding a solution that simultaneously satisfies the constraint equations and extremizes the objective function is, in general, not a straightforward procedure. To this end, many methods of extremizing functions have evolved.

Problems can be divided into two broad categories, linear problems and nonlinear problems. The linear problems have a linear objective function and linear constraint equations which can be expressed in the following general form:

$$\text{extremize } f(x_1,x_2,\ldots,x_n) = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

subject to

$$\sum_{j=1}^{n} a_{ij}x_j - b_i \leqslant 0 \quad , \quad i = 1, \ldots, k$$

$$\sum_{j=1}^{n} a_{ij}x_j - b_i = 0 \quad , \quad i = k+1, k+2, \ldots, h$$

$$\sum_{j=1}^{n} a_{ij}x_j - b_i \geqslant 0 \quad , \quad i = h+1, \ldots, m$$

If the variables appear nonlinearly in either the constraints or the objective function, then the problem is considered to be nonlinear and, consequently, is generally more difficult to solve. Within this class of problems consider these two subclasses: those where the nonlinearities appear only in the objective function, and those where the nonlinearities can appear in both the objective and the constraint equations.

This paper addresses the problem described by a nonlinear objective function and linear constraints. The solution technique is based on a method proposed by Rosen (1960) [1] called the Gradient Projection

Method. In this method, if any of the constraint equations are violated during the unidirectional search, a projection method is used to generate a new feasible direction of search. The functions considered here will be convex within the region of interest. The functions will also be of class C^2 (first and second order partials exist and are continuous). The feasible region is defined by a convex polyhedron formed by the linear constraint equations. Rosen's method, unfortunately, requires the inverse of the matrix formed from the normals of the binding constraints. Binding constraints are the equality constraints and the violated inequality constraints that are treated as equality constraints. Rosen attempted to reduce the complexity of this problem by updating the inverse matrix with a recursive method rather than recalculating it whenever there was a change in the binding constraint set. This recursive method depends on the knowledge of $(N^TN)^{-1}$, where N is the matrix of the normals of the binding constraints and N^T is the matrix transpose [1]. Only one hyperplane can be subtracted or added to the projection matrix[1] at a time. Table 1 shows the number of computations needed to update the projection matrix for the subtraction of one constraint by the Rosen's method and the method proposed in this paper. This table does not include the computations that would be required by the calculation of $(N^TN)^{-1}$. The dimension of the projection matrix calculated by either method is equal to the number of independent variables. The Rosen method does not calculate the projection matrix with matrices of this dimension. The rank of the matrices of this method varies with the number of independent binding constraints. The proposed method saves operations by taking advantage of the matrix symmetry. The last column in Table 1 shows the ratio of the number of operations required by the two methods. This paper proposes a solution technique that does not require the calculation of the inverse of the matrix of the normals of the binding constraints and, for a large number of problems, requires a smaller number of computer operations than does Rosen's method. The proposed method provides similar results with fewer computations, thus in general reducing the computer time. This method is able to add or remove many constraints at a time which the Rosen method does not. A comparison between the two methods is seen in Figure 1 for the case where there are 20 independent variables.

Section II of this paper discusses the general nonlinear optimization problem subject to linear constraints. In Section III the problem of finding an initial feasible point is addressed. Phase I of the Two-Phase Simplex Method is used to provide a feasible point when a user-provided starting point is infeasible or when the problem includes equality constraints, since these must always be satisfied. Section IV discusses a technique for locating an extrema within unconstrained feasible space. The method is the Davidon-Fletcher-Powell (DFP) which is a variable metric technique [2]. Section V analyzes the solution to the problem with constrained extremum and presents the main thrust of the paper. The application of the gradient projection method to locating constrained extrema will be discussed. The differences between the method proposed in this paper and Rosen's method will also be detailed. Section VI contains the conclusions. Appendices A and B provide the user's guide and the results of the test cases to which the program was applied. Appendix C contains a description of the program used in this paper and Appendix D contains the program listing.

II. STATEMENT OF THE PROBLEM

Let $x_i, i=1,2,...,n$, be the coordinates of the point x in n-dimensional Euclidean space. Points in space will be defined with superscripts, and elements of points will be defined with subscripts. Vectors will be represented by columns of elements as

1. An n x n real matrix P is called a projection matrix if and only if $P = P^T$ and $PP = P$.

TABLE 1. COMPUTATION COMPARISON FOR UPDATING PROJECTION MATRIX

Variables	Binding Constraints	Total Number of Operations		
		Rosen	Fox	Ratio
10	2	707	55	12.9
	4	1,167	410	2.8
	6	1,787	710	2.5
	8	2,567	1,010	2.5
20	2	2,512	210	12.0
	4	3,832	1,530	2.5
	6	5,472	2,640	2.1
	8	7,432	3,750	2.0
	10	9,712	4,860	2.0
	12	12,312	5,970	2.1
	14	15,232	7,080	2.2
	16	18,472	8,190	2.3
	18	22,032	9,300	2.4
40	4	13,662	5,840	2.3
	8	24,062	14,240	1.7
	12	37,022	22,640	1.6
	16	52,542	31,040	1.7
	20	70,622	39,440	1.8
	24	91,262	47,840	1.9
	28	114,462	56,240	2.0
	32	140,222	64,640	2.2
	36	168,542	73,040	2.3

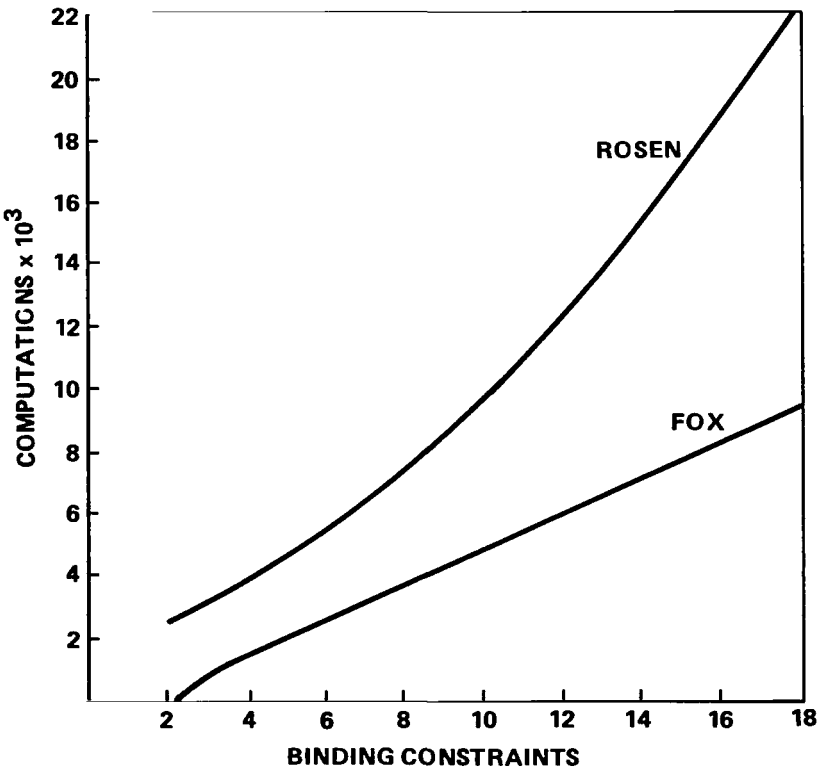


Figure 1. Operations to update the projection matrix.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

x^T will be defined as the transpose of the x vector. In this notation, the general maximizing nonlinear programming problem with linear constraints can be expressed as:

$$\text{max: } f(x) = f(x_1, x_2, \ldots, x_n),$$

subject to the linear equalities and inequalities of the form:

$$(a^i)^T x - b_i = 0 \quad , \quad i = 1,2,\ldots,k<n$$

$$(a^i)^T x - b_i \geqslant 0 \quad , \quad i = k+1,\ldots,p \quad .$$

The vectors a^i, i=1,2,...,p, provide the constraint equation coefficients and the b_i, i=1,2,...,p are scalars. The symbol g_i will represent the ith constraint. These constraints restrict the solution to k hyperplanes and p-k half spaces. Their intersection, R, is a convex polyhedron called the feasible region. R consists of all the points that lie on the equalities and within the half spaces. Points in the equality constraints will lie on the boundary of this region. The constraints are assumed to be linearly independent. The problem is depicted in Figure 2. The curves lines in Figure 2 are the level contours of f(x). This figure shows the linear constraints $g_2, g_3, g_4 \leqslant 0$ and $g_1, g_5 \geqslant 0$ which outline a feasible space that contains x^*. Starting at the unconstrained point x°, the search follows the gradient of $f(x^\circ)$ until a constraint is encountered at x^1. The projection search vectors p^1 and p^2 are calculated at x^1 and x^2, respectively, as new constraints are encountered at these points. A constrained optimum may be found at points that are not located at vertices in nonlinear problems. Inspection of this figure shows that the optimum point x^* occurs on constraint g_3, away from a constraint vertex.

III. FINDING A FEASIBLE POINT

Most iterative search methods require an initial starting point. These points must either be supplied by the user or be generated by an initial point algorithm in the method. For linearly constrained problems the Simplex method can be used to find an initial feasible point if one exists. Many nonlinear methods are developed based on a quadratic function and extended to the general nonquadratic case by approximations of the quadratic. For initial points which are not in the vicinity of the optimal solution, the function may not be adequately represented by a quadratic approximation and the solution algorithm generally takes longer to converge. If the user supplied initial point for linear constrained problems is not in R, then one method of obtaining an initial feasible point is to use Phase I of the Simplex method. The initial point will then lie on the boundary of R at one of the vertices formed by a subset of the constraints, and will automatically satisfy the equality constraints if they are consistent.

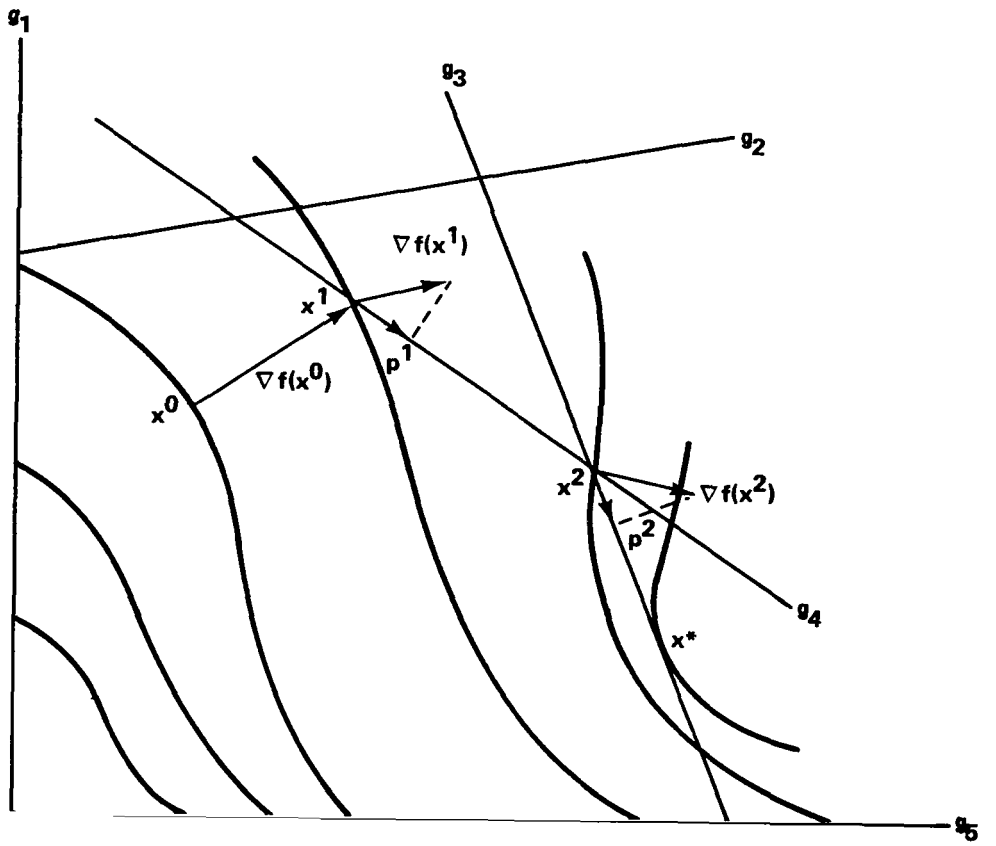


Figure 2. Nonlinear projection optimization.

The Simplex method extremizes a linear objective function subject to the constraint set

$$Ax = b \tag{1}$$

and

$$x_i \geqslant 0 \quad , \quad i = 1,2,...,n$$

where A is an m x n matrix and b is an m-dimensional vector.

The null space of an n x s matrix C is the subspace of all s-dimension vectors y such that

$$Cy = 0$$

Denote the subspace of the vectors that have this property by N(C). If x and y are members of this set, then

$$C(x+y) = Cx + Cy = 0 \quad ,$$

and if k is a scalar then

$$C(kx) = k(Cx) = 0 \quad .$$

Let N(C) be the null space of C and have dimension q. Then the set has q linearly independent vectors which form the basis for the set defined by C. The column rank of a matrix is equal to the number of columns minus the null space dimension. Let the basis of the null space of C be

$$v^1, v^2, \ldots, v^q \quad .$$

This set can be extended to a basis for an s-dimensional space by adjoining w^j, j=1,2.,,,,.r vectors that are linearly independent

$$v^1, v^2, \ldots, v^q, w^1, \ldots, w^r$$

where

$$q+r = s \quad .$$

Now every column vector x in C can be written

$$x = \sum_{i=1}^{q} a_i v^i + \sum_{j=1}^{r} b_j w^j$$

Therefore,

$$Cx = C \sum_{i=1}^{q} a_i v^i + C \sum_{j=1}^{r} b_j w^j = \sum_{i=1}^{q} a_i (Cv^i) + \sum_{j=1}^{r} b_j (Cw^j)$$

Since $Cv^i = 0$ then

$$C_x = \sum_{j=1}^{r} b_j (Cw^j)$$

Hence, the r vectors Cw^j span the co-domain. These vectors are also linearly independent. If

$$k_1(Cw^1) + \ldots k_r(Cw^r) = 0 \quad ,$$

then

$$Ch = 0 \quad ,$$

where

$$h = k_1 w^1 + \ldots + k_r w^r \quad . \tag{2}$$

So h belongs to N(C) and is a linearly independent combination of the r vectors

$$h = \sum_{i=1}^{q} 1_i v^i + \sum_{j=1}^{r} k_j w^j \quad ,$$

a contradiction.

The Simplex method of finding an optimal solution is an iterative update process that solves the constraint equations for their vertices. The direction of the move from one vertex to the next is done in a manner that increases or decreases the objective function depending on whether it is a minimization or maximization problem. Letting z_o be the value of the objective function for the previous iteration,

$$z = z_o + f \text{ (optimizing vector)} \quad .$$

The optimizing vector transforms the last value of the objective function z_o to a new value z that is closer to the optimal value. Let A be partitioned into matrices B and N such that

$$A = (B,N)$$

where B is an m x m invertible matrix and N is an m x (n-m) matrix and the rank of A(B,N) is m. Let

$$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}$$

where the partitions of x have the dimensions m and n-m, respectively, and where x_B is associated with the B partition of A and x_N with the N partition of A. From (1)

$$Ax = (B,N)x = Bx_B + Nx_N \quad .$$

A basic solution will be given by x if

$$x_B = B^{-1}b$$

$$x_N = 0 \quad .$$

If $x \geqslant 0$, then this represents a solution that satisfies all constraints and is called a basic feasible solution (BFS). In general, only a small portion of the vertices will have to be examined in order to locate the optimal solution to a linear objective function using the Simplex algorithm. The Simplex method requires an initial feasible point. Given this, each succeeding vertex will also be feasible. When a better objective function value cannot be found at any adjacent vertex and all $x_{Bi} \geqslant 0$, i=1,2,...,m, an optimum has been achieved. Consider the following minimization problem:

$$\min z = c^T x$$

such that

$$Ax = b$$

$$x \geqslant 0 \quad .$$

Given a basic feasible solution x_{Bf},

$$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$$

let c_B be the coefficients in the objective function associated with the x_B vectors and c_N with the x_N vectors. The objective function z can be given by

$$z = c^T x = (c_B^T, c_N^T) \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix} = c^T B^{-1} b \quad .$$

Let

$$x^T = (x_B, x_N)^T$$

be an arbitrary basic feasible solution. Then

$$x_B \geqslant 0 \quad , \quad x_N = 0 \quad ,$$

and

$$Ax = Bx_B + Nx_N = b$$

Premultiplying by B^{-1} and rearranging

$$x_B = B^{-1}b - B^{-1}Nx_N$$

or

$$x_B = B^{-1}\, b - \sum_{j \epsilon D} B^{-1} a^j x_j \tag{3}$$

where a^j is the jth column vector in A and where D is the current set of indices of the nonbasic variables. The objective function is

$$z = c^T x = c_B^T x_B + C_N^T x_N = c_B^T (B^{-1}b - \sum_{j \epsilon D} B^{-1} a^j x_j) + \sum_{j \epsilon D} c_j x_j$$

Let z_o represent the objective function at some arbitrary basic feasible solution. Then

$$z = z_o - \sum_{j \epsilon D} (z_j - c_j) x_j \quad , \tag{4}$$

where $z_j = c_B{}^T B^{-1} a^j$ for each nonbasic variable. Since z is to be minimized, it can be seen from (4) that whenever $z_j - c_j > 0$ the objective function decreases by introducing some nonbasic variable into the basis at a positive value. This rate of decrease will be the largest if we pick the most positive of these $z_j - c_j$, say $z_k - c_k$. From (3)

$$x_B = B^{-1}b - B^{-1}a^k x_k = \underline{b} - y^k x_k \quad ,$$

where $\underline{b} = B^{-1}b$ and $y^k = B^{-1}a^k$. Then as x_k increases positively from zero, the basis is modified as shown in vector form

$$\begin{bmatrix} x_{B1} \\ x_{B2} \\ \vdots \\ x_{Bm} \end{bmatrix} = \begin{bmatrix} \underline{b}_1 \\ \underline{b}_2 \\ \vdots \\ \underline{b}_m \end{bmatrix} - \begin{bmatrix} y_1{}^k \\ y_2{}^k \\ \vdots \\ y_m{}^k \end{bmatrix} x_k \quad . \tag{5}$$

If $y_i^k \leqslant 0$ i=1,2,...,m, then x_{Bi} increases as x_k increases indefinitely. If all $y_i^k \geqslant 0$ then x_k can increase only until one of the $x_{Bi} = 0$ due to the $x \geqslant 0$ constraints. In the absence of degeneracy (degenerate solutions are those where the value of at least one of the basic variables equals zero), $b_r \geqslant 0$, r=1,2,...,m and then $x_k = \underline{b}_r / y_r{}^k \geqslant 0$. From (4)

$$z = z_o - (z_k - c_k)x_k \quad . \tag{6}$$

From the choice of $z_k - c_k \geqslant 0$ for a minimization problem,

$$z < z_o \quad .$$

Substituting (6) into (5)

$$x_B = \underline{b}_i - (y_i^k / y_r^k)\underline{b}_r \quad , \quad i = 1,2,...,m \quad .$$

All other $x_j = 0$. From above, $x_{Br} = 0$ becomes a nonbasic variable, and $x_k = \underline{b}_r / y_r{}^k$ has been added to replace it, forming a new BFS. This new BFS will decrease the value of the objective function and will also satisfy the constraints.

Phase I of the Simplex method requires the problem to be set up in the standard form [3]. This is accomplished by the use of slack and artificial variables. Standard form is obtained when all variables are

non-negative, all of the constraints are equations, and all constants on the right-hand side are non-negative. In this form, row column operations can be performed. In some problems, the coefficient matrix A may not contain a full unit matrix of the same order as the number of constraints. This occurs when there are equality and less than or equal to constraints. In this case no BFS can be obtained. Artificial variables can be added to the constraints to augment A such that a BFS exists. Non-negative slack variables are introduced as necessary to transform the set of less than or equal to constraints into equalities. This set is then examined to determine if a full unit matrix is present. If it is not, non-negative artificial variables are added to the appropriate equations to create a full unit matrix. The Simplex method begins at the origin of the primary decision variables with all of the vector b allocated to the artificial and slack variables. These artificial variables must be driven to a zero value in the final solution. In Phase I of the two-phase Simplex method, the objective function is replaced with an objective function that sums the artificial variables. This objective function will be minimized. Consider the following set of constraints:

$$x_1 - 2x_2 + x_3 \leqslant 11$$

$$-4x_1\, x_2 + x_3 \geqslant 3$$

$$2x_1 - x_3 = -1$$

$$x_i \geqslant 0 \qquad i = 1,2,3 \quad .$$

The addition of non-negative slack variables x_4 and x_5 allows the problem to be expressed in standard form:

$$x_1 - 2x_2 + x_3 + x_4 = 11$$

$$-4x_1 + x_2 + 2x_3 - x_5 = 3$$

$$-2x_1 + x_3 = 1$$

$$x_i \geqslant 0 \qquad i = 1,2,3,4,5 \quad .$$

Artificial variables can be added to the last two equations to produce the required unit matrix. The Phase I Simplex method will reduce these artificials to a value of zero, if a BFS to the problem exists, and thus remove them from the solution. From the previous example, after adding the artificial variables x_6 and x_7, the Phase I problem is

$$\text{Min } z = x_6 + x_7 \quad ,$$

subject to

$$x_1 - 2x_2 + x_3 + x_4 = 11$$

$$-4x_1 + x_2 + 2x_3 - x_5 + x_6 = 3$$

$$-2x_1 + x_3 + x_7 = 1$$

$$x_i \geqslant 0 \qquad i = 1,2,3,4,5,6,7 \quad .$$

This problem can be represented in the following tableau form:

	c_j	o	o	o	o	o	1	1	
c_B	Basis	x_1	x_2	x_3	x_4	x_5	x_6	x_7	b
0	x_4	1	-2	1	1	0	0	0	11
1	x_6	-4	1	2	0	-1	1	0	3
1	x_7	-2	0	1	0	0	0	1	1
	$\bar{c}$ row	6	-1	-3	0	1	0	0	

The tableau is a representation of the problem in detached coefficient form. The c_B column is made up of the objective function coefficients of the present basic variables listed in the basis column. The coefficients of the objective function are shown in the c_j row. The columns contain all of the coefficients associated with the variables with the exception of the $\bar{c}$ row. The columns that contain the present basic variables also contain the unit matrix. The bottom row is called the cost row and $\bar{c}_j = c_j - (c_B a_j)$. The b column represents the value of the basic variables listed on the same row. The program presented in the last section of the paper addresses the problem of finding an initial feasible point. The most efficient way to accomplish this is through the user's organization of the constraints and variables. Nonstandard forms can usually be put into standard form by simple substitutions or transformations. For example, in the case where

$$-\infty < x_i < \infty \quad ,$$

the substitution of

$$x_i = x_i^+ - x_i^- \quad ,$$

where

$$x_i^+, x_i^- \geqslant 0$$

can be made. The scalar expression

$$|ax| \leqslant b$$

can be expanded in the form

$$-b \leqslant ax \leqslant b$$

which can be used in the form

$$ax \leqslant b$$

$$-ax \leqslant b \quad .$$

Many other nonstandard forms can be standardized with similar substitutions or transformations.

IV. UNCONSTRAINED OPTIMIZATION

Section III discussed the problem of finding an initial feasible point. An initial feasible point is essential to using the method proposed by this paper. Once a feasible point is found, a nonlinear optimization method is then applied to the problem. The method proposed here uses the DFP method when the present point is not located in any constraint hyperplanes and does not violate any of the constraints.

The advantage of combining the Rosen gradient projection method (1960) [1] with the DFP method first introduced by Davidon (1959) and later modified by Fletcher and Powell (1963) [2] is seen in the rate of convergence to the optimal solution. The DFP method, originally developed as a nonlinear unconstrained technique [4,5,6], was adapted to a linear constraint method by Goldfarb (1969) [7]. The DFP method is considered to be one of the most powerful of the nonlinear search techniques and was developed to solve the quadratic function. The method becomes iterative when extended to nonquadratic nonlinear functions. By using the Taylor series, it is possible to determine the value of a function in the neighborhood of a known point. As the neighborhood becomes small, the Taylor series approximates a quadratic function. The smaller the neighborhood the better the approximation. The DFP method combines the best features of the steepest descent technique with those of the Newton method and with few of their drawbacks [8]. The steepest descent technique generates a search vector using knowledge of the first partials of the objective function. The negative gradient of the objective function is represented by the vector

$$-\nabla f(x) = -\begin{bmatrix} \dfrac{\partial f(x)}{\partial x_1} \\ \vdots \\ \dfrac{\partial f(x)}{\partial x_n} \end{bmatrix} \tag{7}$$

and points in the direction of the most rapid decrease of the objective function from the point x. Steepest descent converges to the optimum solution slowly when the function is highly nonlinear. A Newton method converges in a single step when operating on a quadratic objective function. This method requires that the

matrix of second partials be calculated and inverted at each iteration. The DFP method approximates the Hessian inverse using only first partials, and improves this approximation at each step. This method generates the inverse Hessian after n steps in an n-dimensional quadratic problem. Like the method of conjugate gradients, the algorithm of the variable metric DFP method is designed to extremize the following function about an arbitrary point x^o:

$$f = 1/2(x-x^o)^T A(x-x^o) \quad , \tag{8}$$

by conducting a sequence of one-dimensional line searches that begin at an arbitrary point x_i and locate improved points by the relation

$$x^{i+1} = x^i + \alpha_i p^i \quad , \tag{9}$$

where α_i is a positive scalar constant and p^i is a search vector. The constant α_i is chosen so that x^{i+1} is located at the extremum along the direction of search. Since this extremum is located at a point x^{i+1} where the gradient of the objective function is perpendicular to the search vector, their dot product is zero [8]. Let

$$y(\alpha_i) = f(x^i + \alpha_i p^i)$$

$$\partial y(\alpha_i)/\partial \alpha_i = (p^i)^T \nabla f(x^{i+1}) \tag{10}$$

$$= 0 \quad .$$

The extremum along the direction of search occurs at the point x^{i+1} where the search vector is tangent to a contour of the objective function as shown in Figure 3. From (8),

$$\nabla f(x^i) = A(x^i - x^o) \quad . \tag{11}$$

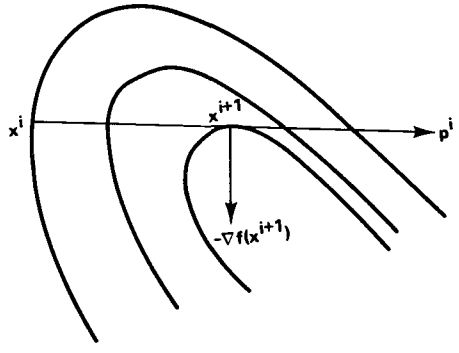


Figure 3. Search vector and gradient at x^{i+1}.

Using (9) recursively yields

$$x^n = x^i + \sum_{j=1}^{n-1} \alpha_j p^j \quad . \tag{12}$$

Subtracting x^o from (12) and premultiplying by A,

$$A(x^n - x^o) = A(x^i - x^o) + \sum_{j=1}^{n-1} \alpha_j A p^j \quad . \tag{13}$$

Then from (11) and (13)

$$\nabla f(x^n) = \nabla f(x^i) + \sum_{j=1}^{n-1} \alpha_j A p^j$$

or as a special case

$$\nabla f(x^{i+1}) = \nabla f(x^i) + \alpha_i A p^i \quad . \tag{14}$$

If a p^i can be chosen so that it is A conjugate to all other search vectors, i.e.,

$$(p^i)^T A p^j = 0 \quad , \quad i \neq j \quad , \tag{15}$$

then two important results can be concluded. First, all of the search vectors p will be linearly independent, and second, the quadratic form will be extremized in no more than n steps in n-dimensional space [9]. Define

$$p^i = -H \nabla f(x^i) \quad , \quad i = 0,1,\ldots,n-1 \quad , \tag{16}$$

where H is an n x n symmetric positive definite matrix. Fletcher and Powell [2] proved that the iterative search method developed by Davidon satisfies the condition specified in (15).

This method will be developed and some of its interesting properties discussed. H will always be symmetric and positive definite if H_o is chosen to be symmetric and positive definite and if H_{i+1} is calculated by

$$H_{i+1} = H_i + B_i + C_i \quad , \tag{17}$$

where B_i and C_i are n x n symmetric matrices that are calculated at each step. Using (17) recursively,

$$H_n = H_o + \sum_{j=0}^{n-1} B_j + \sum_{j=0}^{n-1} C_j \quad . \tag{18}$$

By choosing

$$\sum_{j=0}^{n-1} B_j = A^{-1}$$

and

$$\sum_{j=0}^{n-1} C_j = -H_o \quad ,$$

then (18) reduces to $H_n = A^{-1}$. H_o is usually chosen to be I to assure that it is positive definite and symmetric. The construction of B shows the importance of the A conjugacy as expressed in (15). The individual B_j's in (18) can be determined by this A conjugate condition. Let T be an n x n matrix with columns consisting of the search vectors p^i, i=0,1,...,n-1. Then

$$T^T A T = D \quad , \tag{19}$$

where D is an n x n diagonal matrix. This follows from (15) since

$$d_{ij} = (p^i)^T A p^j = 0 \quad , \quad i \neq j \quad .$$

The double subscripting of a variable will be taken to denote a matrix element. Since A is positive definite, A^{-1} exists and A can be solved from (19)

$$A = (T^T)^{-1} D T^{-1} = (T D^{-1} T^T)^{-1} \quad .$$

Hence

$$A^{-1} = TD^{-1}T^T$$

and thus,

$$\sum_{j=0}^{n-1} B_j = TD^{-1}T^T \quad .$$

From linear algebra, the diagonal terms of D^{-1} are $1/d_{ii}$. Therefore,

$$\sum_{j=0}^{n-1} B_j = \sum_{j=0}^{n-1} (1/d_{jj})\, p^j (p^j)^T \qquad (20)$$

where

$$d_{jj} = (p^j)^T A p^j$$

Substituting this last expression into (20) gives

$$B_j = (p^j (p^j)^T)/((p^j)^T A p^j) \qquad (21)$$

and substituting (14) into the denominator,

$$\sum_{j=0}^{n-1} B_j = \sum_{j=0}^{n-1} \alpha_j (p^j (p^j)^T)/((p^j)^T (\nabla f(x^{j+1}) - \nabla f(x^j))$$

or, in corresponding terms, let

$$B_i = \alpha_i (p^i (p^i)^T)/((p^i)^T (\nabla f(x^{i+1}) - \nabla f(x^i))) \qquad (22)$$

for i=0,1,...,n-1. In a similar manner, C_i will be developed. Post-multiplying (17) with Ap^i, we have

$$H_{i+1}Ap^i = H_iAp^i + B_iAp^i + C_iAp^i \quad .$$

Substituting (21) into the middle right-hand term,

$$H_{i+1}Ap^i = H_iAp^i + (p^i(p^i)^TAp^i)/((p^i)^TAp^i) + C_iAp^i$$

$$= H_iAp^i + p^i + C_iAp^i \quad . \tag{23}$$

If C_i is chosen so that $H_{i+1}Ap^i = p^i$, then p^i is an eigenvector of $H_{i+1}A$. Fletcher and Powell used this property to show quadratic convergence [2]. Using this result in (23)

$$C_iAp^i = -H_iAp^i \quad .$$

Since $(p^i)^TA^TH_iAp^i$ is a scalar value, then

$$1 = ((p^i)^TA^TH_iAp^i)/((p^i)^TA^TH_iAp^i) \quad .$$

Hence,

$$C_iAp^i = -H_iAp^i((p^i)^TA^TH_iAp^i)/((p^i)^TA^TH_iAp^i)$$

or

$$C_i = -(H_iAp^i(p^i)^TA^TH_i^T)/((p^i)^TA^TH_iAp^i) \quad .$$

The last term in the numerator is a result of the symmetry of H. Again, using (14)

$$C_i = -(H_i(\nabla f(x^{i+1}) - \nabla f(x^i))(\nabla f(x^{i+1}) - \nabla f(x^i))^TH_i^T)/((\nabla f(x^{i+1}) - \nabla f(x^i))^TH_i(\nabla f(x^{i+1}) - \nabla f(x^i))) \quad .$$

The third term in the numerator follows from the symmetry of A and from the linear algebra identity

$$p^TA^T = (Ap)^T \quad .$$

The search algorithm of (9) with the a search vector defined by the relationship

$$p^i = -H_i \nabla f(x^i) \quad ,$$

where H_i is updated according to (18) and where the exact step α_i is found along each search vector, will converge to the extremum of a quadratic in no more than n steps in n-dimensional space. Examination of the Taylor's series expansion of a nonquadratic nonlinear function around the optimal solution shows that as the solution approaches the optimal solution, the function becomes dominated by the quadratic terms. In the program presented in this paper, the constant α_i is only an approximation of the exact distance from x^i to the extremum along p^i. The usual method for calculating α_i is to curve-fit the points that bracket or lie in the vicinity of this extremum. This program uses a quadratic fit. No effort was made to determine the consequences of this error on the rate of convergence. The H matrix is reset to I, since H may tend to accumulate errors due to numerical truncation and also due to the errors caused by not finding the exact extremum point along each search vector. In this latter error source, the slight inaccuracy causes the dot product between the search vector and the gradient of the function at that point to differ from zero, and it is also generally necessary to use double precision to reduce the truncation problem. These errors will accumulate in the H matrix at each update. A decision must be made to reinitialize the H matrix after a number of updates. The usual practice is to reset the H matrix to I after n steps in an n-dimension space. The first search vector after this update is the negative gradient of the objective function. Hence, after every n steps when H = I, there is a search taken in the direction of steepest descent.

The advantages of the DFP method lie in the convergence properties that approximate the Newton method while using only first order information similar to the steepest descent method. H only approximates A^{-1} closely after n steps. The combining of the unconstrained search with the projection technique by Rosen allows a search of the feasible region R. Unlike linear programming methods, which are constrained to examine only the vertices, this method is free to traverse interior feasible space.

V. CONSTRAINED OPTIMIZATION

Section IV discussed a method of nonlinear optimization for problems that have no constraints. Many nonlinear problems do have linear constraints, however, and this section will address this type of problem.

The projection method is used at the boundaries of a convex polyhedron where the extremum lies outside or on the boundaries of the feasible region R. This method is based on the projection of the gradient of a nonlinear function onto these boundaries [10]. The space E^m is defined as a Euclidean space of dimension m and contains a subspace that is spanned by the normals of q binding constraints. The normals of these binding constraints are the column vectors of a matrix N_q, where the subscript q denotes that the matrix is an m x q matrix. The symbol g_i will designate the ith hyperplane associated with the ith constraint. The matrix

$$(N_q{}^T N_q)^{-1} N_q{}^T$$

is used as pseudo-inverse of the matrix N_q. Let a set of q linearly independent unit vectors u^i, i=1,...,q $\leq$ m, be normals to a set of q linearly independent hyperplanes. Let

$$N_q = (u^1, u^2, ..., u^q) \quad . \tag{24}$$

Because of the linear independence of the q vectors, the q x q symmetric matrix $N_q{}^T N_q$ is nonsingular, and therefore its inverse $(N_q{}^T N_q)^{-1}$ exists. Let Q be a subspace of E^m that is spanned by the vectors $u^1, u^2, ..., u^q$. Let V be the orthogonal complement of Q in E^m. Then V is a subspace of dimension (m-q). Rosen's projection method uses a matrix given by

$$\tilde{P}_q = N_q(N_q{}^T N_q)^{-1} N_q{}^T \tag{25}$$

that projects a general vector in E^m onto Q. To show this, let v ϵ V. Since V is orthogonal to Q,

$$(u^i)^T v = 0 \quad , \quad i = 1, ..., q$$

or

$$N_q v = 0 \quad .$$

Using this in (25),

$$\tilde{P}_q \, v = 0 \quad .$$

In Q let w be a vector, which can then be represented as a linear combination of the q vectors that span Q. Then

$$w = \sum_{i=1}^{q} a_i g^i \quad , \tag{26}$$

and from (24),

$$w = N_q \, a \quad ,$$

where the vector a is the column vector of the coefficients in (26). Therefore,

$$\tilde{P}_q w = N_q (N_q{}^T N_q)^{-1} N_q{}^T N_q a$$

$$= N_q a$$

$$= w \quad .$$

These results show that $\tilde{P}_q$ is a projection matrix that takes any vector in E^m into Q. Let P_q be defined by

$$P_q = I - \tilde{P}_q \quad .$$

If v and w are defined as before, then

$$P_q v = v - \tilde{P}_q v = v - 0 = v$$

and

$$P_q w = w - \tilde{P}_q w = w - w = 0 \quad .$$

From this, any arbitrary vector, $x \in E^m$, decomposed as

$$x = v + w$$

is projected onto the intersection of the q hyperplanes. It is also clear that if q = m, then $P_q = I$ and

$$P_q x = 0 \quad .$$

From vector analysis, the projection of a vector z onto the ith hyperplane where $\nabla n_i(x)$ is the normal to this hyperplane is

$$p^i = z - d_i \nabla n_i(x) \quad .$$

The variable d_i is a scalar constant. Figure 4 depicts this where $z = \nabla f(x)$. If p is specified to be the projection onto Q, then

$$p = \nabla f(x) - \sum_{i=1}^{q} d_i \nabla n_i (x) \quad .$$

The projection p will lie along the intersection of the q hyperplanes.

A Langrangian function L is a scalar function of x and λ where, if λ_i is a scalar multiplier and $h_i(x)$ designates the ith equality constraint, then

$$L(x,\lambda) = \nabla f(x) + \sum_{i=1}^{\delta} \lambda_i \nabla n_i(x) + \sum_{i=s+1}^{\ell} \lambda_i \nabla h_i(x) \quad .$$

At a constrained extremum, the gradient of the Lagrangian vanishes [8,11] and since $\lambda_i = 0$ for nonbinding constraints,

$$\nabla L = \nabla f(x) + \sum_{i=1}^{s} \lambda_i \nabla n_i(x) + \sum_{i=s+1}^{\ell} \lambda_i \nabla h_i(x) = 0 \quad .$$

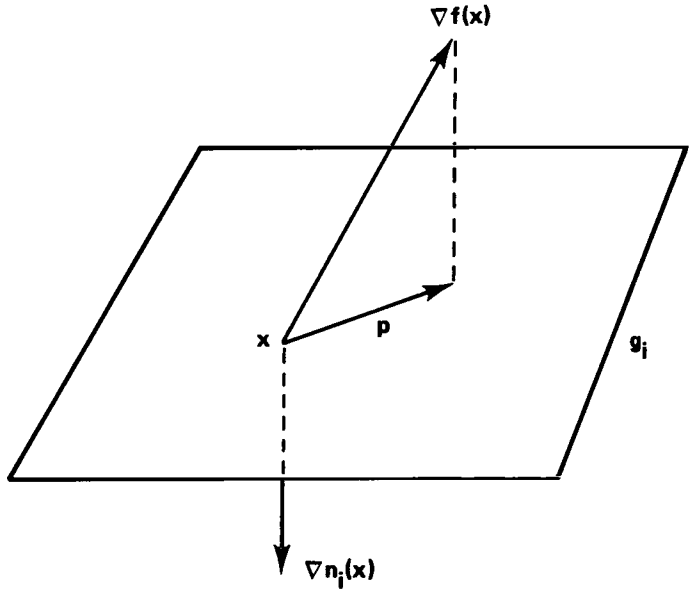


Figure 4. Projection of $\nabla f(x)$ onto hyperplane g_i.

Upon substituting the d_i's for λ_i's, the equivalency between the Lagrangian and the projection equation is obvious. The Kuhn-Tucker [12] conditions show that, if the d_i's have the proper signs, then

$$p = \nabla f(x) - \sum_{i=1}^{s} d_i \nabla n_i(x) - \sum_{i=s+1}^{\ell} d_i \nabla h_i(x) = 0$$

at a bounded optimum. Thus, the projection vector p vanishes at this point. The Kuhn-Tucker conditions are based on Farkas Theorem [12,13] which states that only one of the two following systems has a solution:

System: 1 $Ax < 0$ and $cx > 0$

or

System: 2 $w^T A = c$ and $w > 0$

where A is a given m x n matrix, c is a given vector of dimension n, and x,w are variable vectors of dimension m and n. The first part of the proof of this will be by contradiction. If System 2 has a solution w such that $w^T A = c$ and $w > 0$, let x be such that $Ax < 0$. Then $c^T x = w^T Ax < 0$ which contradicts the given statement. If System 2 has no solution, then $c \notin S = (w^T A : w>0)$. There is some x such that $c^T x > w^T Ax$ for all $w > 0$. If $w = 0$ then $c^T x > 0$, and as w approaches infinity $Ax < 0$ completing the proof. The Kuhn-Tucker conditions follow from this theorem [14]. Let Z be a nonempty open set in E^m, and let $f: E^m \rightarrow E^1$, $n_i: E^m \rightarrow E^1$ for i=1,...,s, and $h_i: E^m \rightarrow E^1$ for i=s+1,...,1. Consider the problem L where $n_i(x)$ is the ith constraint equation.

Min f(s)

such that

$n_i(x) < 0$, $i = 1, 2, ..., s$

$h_i(x) = 0$, $i = s+1, ..., 1$

and

$x \in Z$.

Let x be a feasible solution, and let $I = (i: n_i(x) = 0)$. Suppose that f and n_i for $i \notin I$ are differentiable at x, that n_i for $i \in I$ is continuous at x, and that h_i for i=s+1,...,1 is continuously differentiable at x. Further suppose that $\nabla n_i(x)$ for $i \in I$ and $\nabla h_i(x)$ for i=s+1,...,1 are linearly independent. If x* solves problem L locally, then there exist scalars λ_i for i=1,...,1 such that

$$\nabla f(x^*) + \sum_{i \epsilon I} \lambda_i^* \nabla n_i(x^*) + \sum_{i=s+1}^{\ell} \lambda_i^* \nabla h_i(x^*) = 0$$

and

$$\lambda_i^* \geqslant 0 \quad , \quad i \epsilon I \quad .$$

In addition to the above assumptions, if n_i for $i \notin I$ is also differentiable at x, then the Kuhn-Tucker conditions could be written in the following equivalent form:

$$\nabla f(x^*) + \sum_{i=1}^{s} \lambda_i^* \nabla n_i(x^*) + \sum_{i=s+1}^{\ell} \lambda_i^* \nabla h_i(x^*) = 0$$

$$\lambda_i^* n_i(x^*) = 0 \quad , \quad i = 1,...,s$$

$$\lambda_i^* \geqslant 0 \quad , \quad i = 1,...,s \quad .$$

This shows that $\nabla f(x^*)$ is a linear combination of the normals to the constraints. These normals describe a cone which contains the function gradient. If A is the matrix of the constraints, then by Farkas Theorem, the intersection of the half space defined by $Ax < 0$ and the open half space $cx > 0$ is empty and therefore has no solution. A geometric interpretation is shown in Figure 5. If some $d_i = 0$ or has the wrong sign, then the associated constraint hyperplane is nonbinding and must be dropped from the set used to define the projection matrix. At x^* the gradient of the Lagrangian vanishes when $\nabla f(x^*)$ becomes a linear combination of the binding constraints. The projection scalar d and the Lagrange multiplier λ are equivalent.

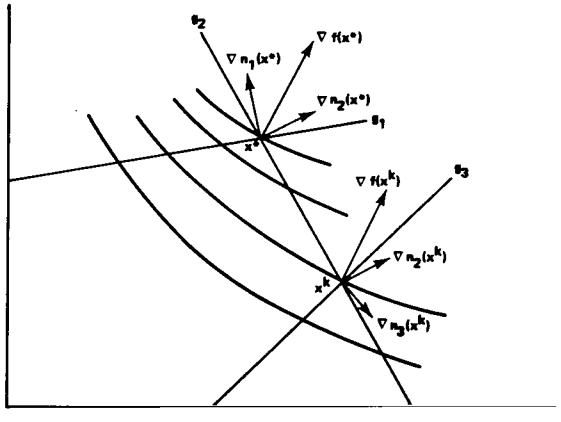


Figure 5. Graphical display of Farkas Theorem.

The number of computations to determine $(N_q{}^T N_q)^{-1}$ is formidable when N_q is very large. If this inverse was recalculated each time a change was made in the defining set of g_i's, the required computer time could become prohibitive. Rosen used a recursive scheme to update the $(N_q{}^T N_q)^{-1}$ matrix rather than recalculating it each time a change was made in the binding constraint set. Although this method greatly reduced the amount of computation necessary to calculate a new inverse, it can remove or add only one constraint at a time.

In this paper, a method is presented that projects an arbitrary vector $x \epsilon E^m$ onto a subspace $Q \subset E^m$ without the necessity of calculating an inverse. For a projection onto a hyperplane, it is necessary to find an exact point in the constraint hyperplane at which to calculate the gradient of the objective function. This gradient will then be projected onto the intersection of the binding constraints. The development of this method follows.

A search along the vector $H^i \nabla f(x^i)$ is made using a scalar multiplier α^i. Let

$$x^{i+1} = x^i + \alpha^i H^i \nabla f(x^i) \tag{27}$$

where the superscripts denote specific iterations. Suppose that at some iteration at least one constraint is violated by the point x^{i+1}. Let this constraint be the jth hyperplane.

$$a_1{}^j x_1{}^{i+1} + a_2{}^j x_2{}^{i+1} + \ldots + a_m{}^j x_m{}^{i+1} = b_j \quad . \tag{28}$$

Substituting (27) into (28) gives

$$a_1{}^j(x_1{}^i + \alpha^i H^i \nabla f(x^i)_1) + a_2{}^j(x_2{}^i + \alpha^i H^i \nabla f(x^i)_2) + \ldots + a_m{}^j(x_m{}^i + \alpha^i H^i \nabla f(x^i)_m) = b_j \quad ,$$

where $H^i \nabla f(x^i)_k$ is the kth element of the search vector. Upon gathering terms

$$(a^j)^T x^i + \alpha^i (a^j)^T H^i \nabla f(x^i) = b_j \quad .$$

Then

$$\alpha^i (a^j)^T H^i \nabla f(x^i) = b_j - (a^j)^T x^i \quad .$$

Since $(a^j)^T H^i \nabla f(x^i)$ is a scalar, α^i can be computed as

$$\alpha^i = (b_j - (a^j)^T x^i)/((a^j)^T H^i \nabla f(x^i)) \quad .$$

This α^i is used to characterize the distance from x^i to a point at the intersection of the search vector and each constraint hyperplane that is violated by x^{i+1}. In the case where x^i is an interior point in feasible space and x^{i+1} is an exterior point, the smallest α^i would be selected. The selected α^i will be denoted as α^*. The knowledge of α^i will be used to eliminate the nonbinding constraints along a search vector. Figure 6 shows graphically that the search vector p^i from x^i may intersect many constraints. Note that g_3 should be the only binding constraint and also has the smallest α^i.

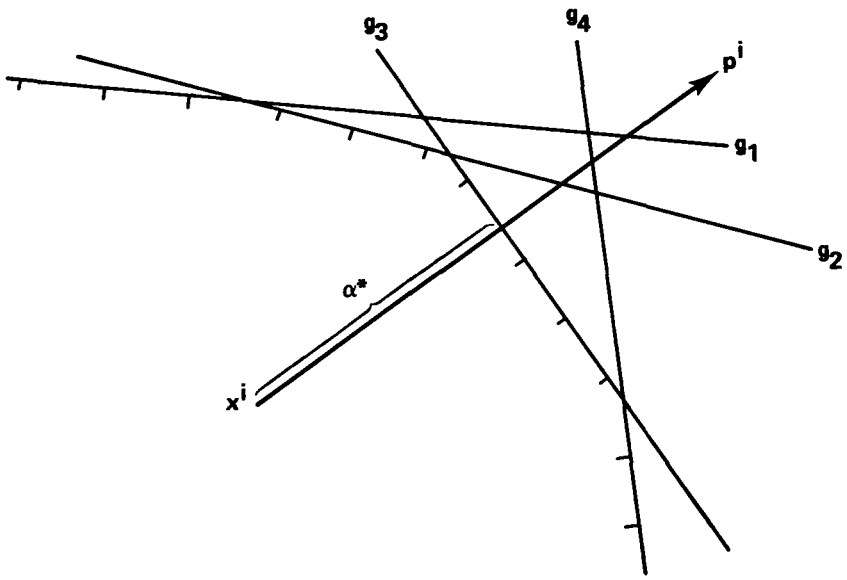


Figure 6. Intersection of search vector and constraints.

The method of calculating the projection matrix was developed from the well-known Gram-Schmidt technique [15]. All the $\nabla n_i(x)$'s of the binding constraints span the q-dimensional space Q. From abstract algebra it is known that the basis of Q is not unique and by employing the Gram-Schmidt method an orthonormal basis for Q can be derived. Choosing g_j as an arbitrary element of the constraint manifold, other elements can be found by the method used by A. O. Morris [16]:

$$
\begin{aligned}
i^1 &= \nabla n_j(x) \\
i^2 &= \nabla n_{j+1}(x) - (\nabla n_{j+1}(x)^T i^1)\, i^1 / \|i^1\|^2 \\
i^3 &= \nabla n_{j+2}(x) - (\nabla n_{j+2}(x)^T i^1)\, i^1 / \|i^1\|^2 - (\nabla n_{j+2}(x)^T i^2)\, i^2 / \|i^2\|^2 \\
&\vdots \\
i^q &= \nabla n_q(x) - (\nabla n_q(x)^T i^1)\, i^1 / \|i^1\|^2 - \ldots - (\nabla n_q(x)^T i^{q-1})\, i^{q-1} / \|i^{q-1}\|^2 \quad .
\end{aligned}
\tag{29}
$$

The i^j vectors are changed to a unit vector by replacing each i^j by its unit vector where

$$u^j = i^j / \|i^j\| \quad .$$

The projection of $\nabla f(x)$, where x is a point in the intersection of the q constraint hyperplanes, onto this vector set will be

$$p = (I - u^1(u^1)^T - u^2(u^2)^T - \ldots - u^q(u^q)^T) \nabla f(x) \quad .$$

Let

$$A = \sum_{j=1}^{q} u^j(u^j)^T \quad .$$

Then

$$p = (I - A) \nabla f(x) \quad .$$

The method of calculating this A matrix is crucial to the efficiency of the method. From (29) it can be seen that the Gram-Schmidt method is also a projection of each succeeding $\nabla n_i(x)$ upon the previous vector set. Using this, the following recursive method was developed to update A:

$$i^{i+1} = (I - A_i) \nabla n_{i+1}(x) \quad . \tag{30}$$

This vector is normalized to a unit vectors as before. Then

$$A_{i+1} = A_i + u^{i+1} (u^{i+1})^T \quad . \tag{31}$$

When updating A by the addition of constraints, they can be added to the A matrix in any order. The removal of constraints is not so simple. In the algorithm, all elements of the A matrix are set to zero and are then recalculated. Some of the previous computational work is saved which allows a portion of the A matrix to be recalculated with very little computation. Each iteration that updates the projection matrix has the same properties as the Gram-Schmidt method in that it is sensitive to the order in which the vectors are used in the algorithm. Because of this, the algorithm does not produce a unique projection matrix at intermediate steps. To eliminate a constraint, simple subtraction cannot be used. The vectors must be removed in the reverse order in which they are added. This continues until all the nonbinding constraints have been removed. This process will also remove some binding constraints which must then be re-added. The same

number of operations are required to remove a vector as to add a vector. If the vector to be removed had been added at the mid-iteration of the algorithm then, as many operations would be required to remove the constraints in reverse order and then regenerate the matrix, as generating the entire projection matrix would require. To minimize the computational effort to regenerate the projection matrix, the program presented saves the u's of the appropriate binding constraints. By zeroing all elements in the projection matrix and updating it with only the active constraints, the effect is the same as that of dropping all the nonbinding constraints at once and adding only the binding constraints.

VI. CONCLUSIONS

The theoretical development for the method of nonlinear optimization presented in Section IV shows the applicability of this method and some of the advantages over the Rosen method. The method proposed by this paper provides similar results with fewer computations per iteration. It has been noted earlier that the method presented here eliminates the need to calculate $(N^TN)^{-1}$ as required by the Rosen method. This not only saves computer time, but avoids a potentially formidable matrix inversion problem as the system becomes large. Another advantage of the method proposed here is that there is no limit to the number of constraints that can be simultaneously added or removed from the binding set. The Rosen method is limited to one addition or removal at a time.

A computer program was developed which implements this method. It is constructed in a modular format to facilitate understanding and provide efficiency in application. The program was applied to a variety of test problems and was successful in finding the optimum in each case. These cases were chosen to evaluate the program's ability to converge while encountering some of the difficulties that are inherent in projection methods. A good comparison between this method and the one used by Rosen could not be made because of possible differences in the type of computer used and the programming structure. The program's computer run time is strongly dependent on the techniques used in the line search and the speed and register capacity of the computer. The cases were chosen not as a comparison to problems worked by other programs, but to demonstrate the abilities of this program. The only direct comparison is based on the number of computations required by each method to update the projection matrix. This should provide some idea of the relative speed of operation during the calculation of the projection matrix. Table 1 in Section I shows that the method proposed by this paper should be more efficient with computer time when calculating the projection matrix.

An area of future research would be an investigation of the differences of the projection matrix used by the two methods. A preliminary look at the projection matrix generated by each method for several cases showed that they were the same. If the two methods do generate the same projection matrix for all cases, this should be proven rigorously.

REFERENCES

1. Rosen, J. B.: The Gradient Projection Method for Nonlinear Programming. Part I. Linear Constraints. Journal of the Society for Industrial and Applied Mathematics, Vol. 8, 1960, pp. 181-217.

2. Fletcher, R. and Powell, M. J. D.: A Rapidly Convergent Descent Method for Minimization. Computer Journal, Vol. 6, 1963, pp. 163-168.

3. Murty, Katta: Linear and Combinatorial Programming. John Wiley and Sons, Inc., New York, 1976.

4. Pierre, Donald A.: Optimization Theory with Applications. John Wiley and Sons, Inc., New York, 1969.

5. Phillips, Don T., Ravindran, A., and Solberg, James J.: Operations Research: Principals and Practice. John Wiley and Sons, Inc., New York, 1976.

6. Pearson, J. D.: Variable Metric Methods of Minimization. Computer Journal, Vol. 6, 1979, pp. 171-178.

7. Goldfarb, Donald: Extension of Davidon's Variable Metric Method to Maximization Under Linear Inequality and Equality Constraints. SIAM Journal of Applied Mathematics, Vol. 17, 1969.

8. Gottfried, Byron S. and Weisman, Joel: Introduction to Optimization Theory. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.

9. Perlis, Sam: Theory of Matrices. Addison-Wesley Publishing Company, Inc., Reading, Mass., 1952.

10. Nering, Evar D.: Linear Algebra and Matrix Theory. Second Edition, John Wiley and Sons, Inc., New York, 1970.

11. Pierre, Donald A. and Lowe, Michael J.: Mathematical Programming Via Augmented Lagrangians. Addison-Wesley Publishing Company, Inc., Advanced Book Program, Reading, Mass., 1975.

12. Bazaraa, Mokhtar S. and Jarvis, John J.: Linear Programming and Network Flows. John Wiley and Sons, Inc., 1977.

13. Bazaraa, Mokhtar, S. and Shetty, C. M.: Nonlinear Programming Theory and Algorithms. John Wiley and Sons, Inc., New York, 1979.

14. Martos, Bela: Nonlinear Programming Theory and Methods. American Elsevier Publishing Company, Inc., New York, 1975.

15. Enslein, Ralston, and Wilf: Statistical Methods for Digital Computers. Volume III, John Wiley and Sons, Inc., New York, 1977.

16. Morris, A. O.: Linear Algebra. Van Nostrand Reinhold Company, New York, 1978.

APPENDIX A. USERS GUIDE

The computer program for the solution method developed in this paper requires two subroutines and a data deck to be supplied by the user. The two subroutines are incorporated into the program as subroutines called FUNCT and GRAD. They take the following form:

```
SUBROUTINE FUNCT(X,F,N1)

DIMENSION X(N1)

F = Objective function

RETURN

END
```

and

```
SUBROUTINE GRAD(X,G,N1,NUM)

DIMENSION X(N1),G(N1)

G(1)=the first gradient element (∂f(x)/∂x1)

.
.
.

G(N1)=the N1th gradient element (∂f(x)/∂xN1)

DO 1 I=1,N1

1 G(I) = NUM*G(I)

RETURN

END
```

where

N1 is the number of independent variables

F is the function to be extremized

X(i) is the ith variable element

G(i) is the ith gradient element of F

NUM=1 for maximization and = -1 for minimization.

The coefficients and right-hand side constrant value for the system of linear constraints is read into the program from the data deck with the initializing data. The description of each of the data cards follows:

DATA CARD 1:

(MAX/MIN),N1,NN,NC,EP1,EP2,NLEC,NEC,NGEC,NGEX

These are read according to FORMAT(A3,3I3,2E14.7,4I3) where

MAX for maximization or MIN for minimization problems

N1 – Number of variables in the problem

NN – Maximum number of unconstrained iterations allowed. This will limit the number of iterations in case of slow convergence to the solution for the unconstrained DFP method.

NC – NLEC+NEC+NGEC+NGEX

EP1 – If $|F(x^{i+1})-F(x^i)| \leqslant$ EP1; EP1 is typically = 0.0001 and detects small changes in the function value. This is a condition for convergence.

EP2 – If $\| \nabla F(x^{i+1}) \| \leqslant$ EP2; EP2 is typically equal to 0.0001 and is the norm of the function gradient. It is also used as a condition for convergence.

NLEC – The number of $\leqslant$ constraints

NEC – The number of = constraints

NGEC – The number of $\geqslant$ constraints (excluding NGEX)

NGEX – The number of $x_i \geqslant 0$ i=1,2,...,n constraints.

DATA CARD 2:

X0(I),I=1,N1

X0 is the initial point and is read according to FORMAT(10E13.6). If NC=0 (unconstrained problem), the following data cards are omitted.

DATA CARD 3:

BC(I),I=1,NC

BC(I) provides the right-hand side of the constraints. The BC(I) values must be provided in the following order (1) less than or equal to constraints, (2) equality constraints, (3) greater than or equal to constraints, and (4) nonnegative variable constraints. This ordering is illustrated by the following example:

$g_i(x) \leqslant b \quad i = 1,2,...,NLEC$

$g_j(x) = b \quad j = 1,2,...,NEC$

$g_k(x) \geqslant b \quad k = 1,2,...,NGEC$

$g_\ell(x) = x_\ell \geqslant 0$

The BCs represent the b values and are read according to FORMAT(10E13.6). The problem requires a standard formulation such that $x_i \geqslant 0$ for all i.

DATA CARDS FOR CONSTRAINTS:

The coefficients for each of the constraints are read according to FORMAT(10E13.6). The order for the constraints must match the order for the right-hand side constants on data card 3. All coefficients must be included, i.e., N1 coefficients for each constraint including zeroes.

Example problem and corresponding user input:

Consider the following example:

$\max F = (x_1 - 3)^2 + 9)x_2 - 5)^2$

X0 = (0.5,-1.2,)

Such that

$2x_1 + x_2 \leqslant 20$

$x_1 + 2x_2 \leqslant 40$

$x_1 + 2x_2 \leqslant 30$

$9x_1 + 6x_2 = 100$

$x_1 \geqslant 0$

$x_2 \geqslant 0$.

SUBROUTINES:

```
SUBROUTINE FUNCT(X,F,N1)
DIMENSION X(N1)
F=(X(1)-3.)**2+9.*(x(2)-5.)**2
RETURN
END
```

```
SUBROUTINE GRAD(X,G,N1,NUM)
DIMENSION X(N1),G(N1)
G(1)=2.*(X(1)-3.)
G(2)=18.*(X(2)-5.)
DO 1 I=1,N1
1 G(I)=NUM*G(I)
RETURN
END
```

DATA CARDS

```
MAX  2  70    6          0.0001          0.0001    3    1    0    2
0.5     -1.2
20.     40.        30.        100.
2.      1.
1.      2.
1.      2.
9.      6.
1.      0.
0.      1.
```

APPENDIX B. TEST PROBLEMS

This section presents the results of several test problems that were solved with the technique developed in this paper. Unconstrained problems were selected to test the DFP method. Linearly constrained problems were then selected to test the projection technique. The program was initialized at infeasible points for some cases to test the algorithm that generates a feasible starting point.

UNCONSTRAINED TEST PROBLEMS AND RESULTS

The following optimization problems were unconstrained. The program uses the DFP method with the approximate Hessian inverse matrix reset to I after N+1 steps. The restrictions for $x_i \geqslant 0$ are relaxes for the unconstrained problems in which NC = 0.

T1: BANANA FUNCTION

$$\text{MIN } f(x) = \sum_{i=1}^{N-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

The starting point with N = 3 is x^o = (-1.,0.,-1.). The minimum was found after 73 iterations of the DFP method. The program execution time was 0.1125 min on a SIGMA-V computer. The minimum point of the objective occurred at x^* = (1.,1.,1.) with $f(x^*) = 0.0$. This problem is also known as Rosenbrock's Function when N = 2. The Banana function is considered to be a severe test for nonlinear optimization algorithms due to the steep valley generated by the coefficient value of 100.

T2: $$\text{MIN } f(x) = \sum_{i=1}^{N-1} [10(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

The starting point with N = 3 is x^o = (6.67,6.67,0.). The minimum of the objective occurred at x^* = (1.,1.,1.) with $f(x^*) = 0.0$. This problem is similar to T1. The difference is the reduced steepness of the valleys formed by the $(x_{i+1} - x_i^2)^2$ terms. Computer time for execution was 0.0284 minutes and the number of iterations was reduced to only 12.

T3: $\text{MIN } f(x) = (x_1 - 3)^2 + 9(x_2 - 5)^2$

The starting point is x^o = (1.,1.) and the minimum occurred at x^* = (3.,5.) with $f(x^*) = 0.0$. Three iterations of the DFP method were performed. The computer time for execution was 0.0159 minutes.

T4: $$\text{MIN } f(x) = \sum_{i=1}^{10} (x - 10)^2$$

The starting point is x^0 = (0.,1.,2.,3.,4.,5.,6.,4.,2.,4.). The minimum occurred at x* = (10.,10.,10.,10.,10., 10.,10.,10.,10.,10.) with f(x*) = 0.0. Only one iteration of the DFP method was required since this is a spherical function. The total execution time was 0.0292 minutes.

CONSTRAINED TEST PROBLEMS AND RESULTS

For these problems the iterations when using a projected search vector are separated from those of the DFP method. The Gradient Projection Method makes no use of the Hessian (or its inverse). For the constrained problems, once the solution sequence encountered the boundary of feasible space, it remained on the constraint manifold. All these examples were initialized outside of feasible space in order to test the routine that finds a feasible initial point.

$$\underline{T5}: \quad \text{MIN } f(x) = \sum_{i=1}^{2} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

subject to

$$2x_1 + x_2 + x_3 \leqslant 20$$

$$x_1 + 2x_2 + 4x_3 \leqslant 40$$

$$x_1 + 2x_2 + 2x_3 \leqslant 30$$

$$9x_1 + 6x_2 + x_3 = 100$$

$$10x_1 + 20x_2 + x_3 \geqslant 100$$

$$x_i \geqslant 0 \quad , \quad i = 1,2,3$$

The minimum of the objective occurred at x* = (6.67,6.67,0.) with f(x*) = .33E+6. The search terminated at this point due to satisfying a program convergence test. Since it is not possible to drop equality constraints from the projection matrix, the last point is taken as a constrained extremum. The minimum was found with two projection iterations and no DFP iterations. The total execution time was 0.0630 minutes.

$$\underline{T6}: \quad \text{MAX } f(x) = \sum_{i=1}^{2} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

The constraints are the same as those in T5. The maximum was found at x* = (1.67,14.17,0.) with f(x*) = .34E+6. At this point a constrained maximum occurred caused by the restriction of not dropping the equality constraints from the projection matrix. This demonstrates the ability to move around on the equality constraint manifold in the search for an extremum. This point was found after three projection iterations and no DFP searches. The execution time was 0.1186 minutes.

T7: Max[T1]

such that

$$2x_1 + x_2 + 4x_3 \leqslant 20$$

$$x_1 + 2x_2 + 4x_3 \leqslant 40$$

$$x_1 + 2x_2 + 2x_3 \leqslant 30$$

$$9x_1 + x_2 + x_3 \leqslant 100$$

$$10x_1 + 20x_2 + x_3 \geqslant 100$$

$$x_i \geqslant 0 \quad , \quad i = 1,2,3$$

The constrained maximum was found at x* = (0.,15.,0.) with the value of f(x*) = .65E+5. The maximum was obtained in one projection search, and the program execution time was 0.0703 minutes.

The results for T5, T6, and T7 are shown in Figure B-1. The tests T5 and T6 have an equality constraint that the extremum is required to satisfy. The equality constraint is constraint number 4. The test T7 had no equality constraints and the extrema for this problem was found at a vertex. The equality constraint was changed to an inequality. The test T5 is a minimization and the test T6 is a maximization of the objective function which explains why the extrema are found as far away from each other as possible and still lie on the equality constraint.

T8: MIN[T7]

The constrained minimum was found at x* = (2.8,3.5,2.7) with f(x*) = .1E+5. The minimum was found after eleven projection iterations. The execution time was 0.1020 minutes. Constraints 1 and 5 are binding as shown in Figure B-2. The initial point was x^0 = (0.,5.,0.). From this point the search proceeded along constraint 5 until constraint 1 was encountered. The minimum of the objective function was found in the seam formed by constraints 1 and 5.

T9: MIN $[f(x) = 5e^{x_1 x_6} + x_2 x_4^3 - x_3 \sin(2(x_7)) + x_5^2 - 10x_8 x_5]$

such that

$$x_2 + 10x_4 + x_8 \leqslant 35$$

$$x_3 - 5x_5 + x_6 - x_7 \leqslant 8$$

$$x_1 + x_3 + x_6 \leqslant 10$$

$$x_1 + x_2 + 3x_4 + x_5 + 7x_7 \leqslant 200$$

$$-x_2 + 15x_3 + x_6 - x_7 + 3x_8 = 25$$

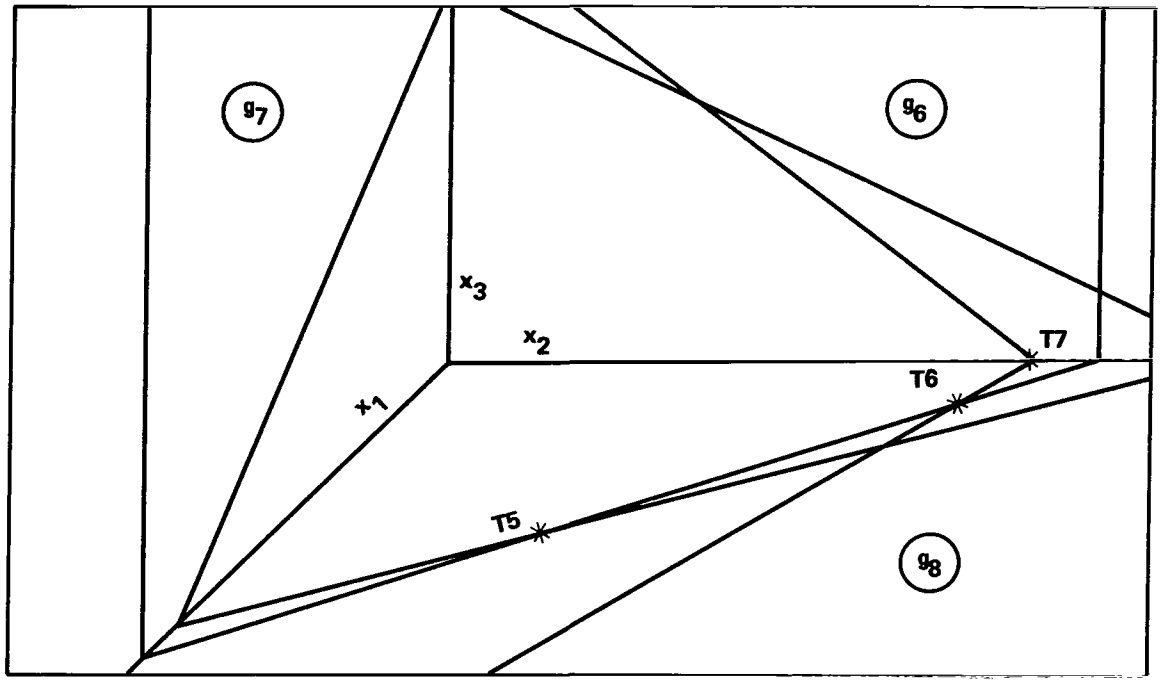


Figure B-1. Optimal points for problems T5, T6, and T7.

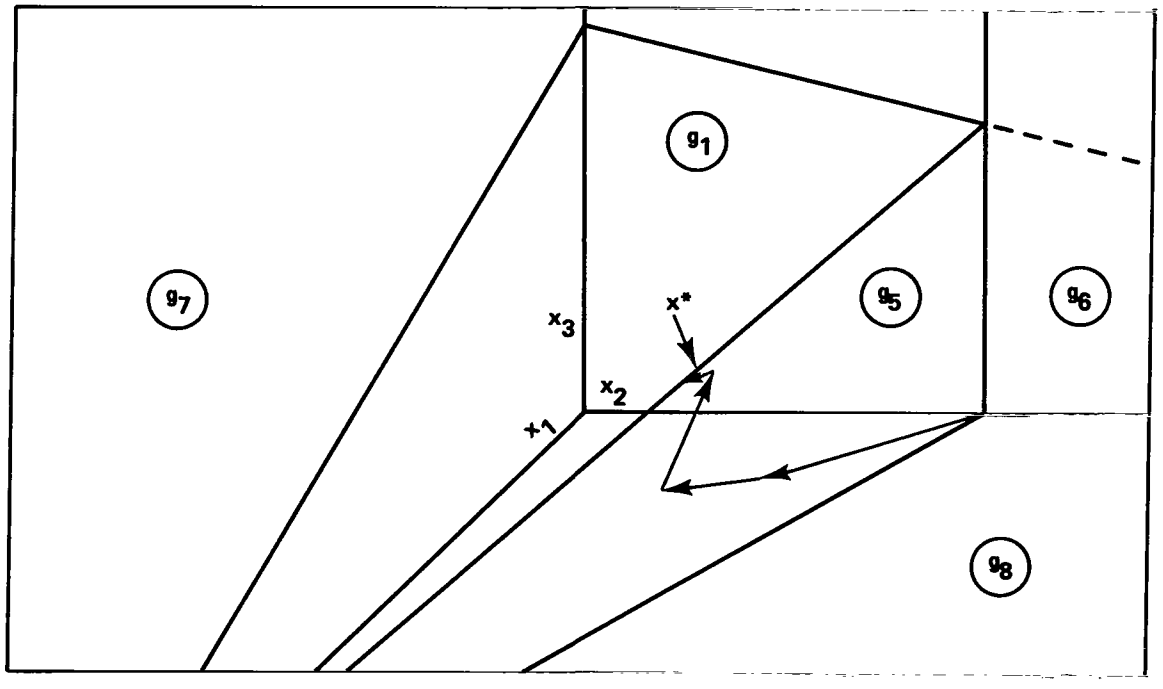


Figure B-2. Optimal point for problem T8.

$$x_4 - x_5 + 5x_6 - 10x_8 = 27$$

$$-3x_2 - 8x_3 + x_5 + 6x_7 \geq 10$$

$$x_1 + 15x_2 + x_5 + 8x_6 \geq 17$$

$$x_1 - 2x_2 - 8x_3 + 5x_4 + x_5 - 7x_6 + 10x_7 + 3x_8 \geq 50$$

$$x_i \geq 0 \quad , \quad i = 1, 2, ..., 8$$

The starting point x^o = (-1.,0.,-1.,0.,-1.,0.,-1.,0.) was outside the feasible region and the program generated a new starting point x^o = (0.,0.,1.89014,3.5,0.,4.7,8.05211,0.). The minimum of the objective function occurred at x^* = (0.,0.,1.89014,3.5,0.,4.7,8.05211,0.) with $f(x^*)$ = 5.727. This constrained minimum was found after six projection iterations. The extremum was found in constraints 1, 5, 6, 9, 10, 11, 14, and 17. The execution time was 0.1224 minutes. This is a vertex of the constraints.

T10: MIN $[f(x) = (x_1 - 18)^2 + (x_2 - 18)^2]$

such that

$$-20x_1 + 4x_2 \leq 20$$

$$-4.7x_1 + 2x_2 \leq 14$$

$$-1.5x_1 + x_2 \leq 11$$

$$-2x_1 + 2x_2 \leq 20$$

$$0.4x_1 + 2x_2 \leq 40$$

$$2x_1 + 0.4x_2 \leq 42$$

$$x_i \geq 0 \quad , \quad i = 1, 2.$$

The starting point was x^o = (0,0) and the minimum point was x^* = (17.54,16.49) which is a vertex. Figure B-3 shows the search path the program followed to acquire the constrained extremum. This shows the advantage of the nonlinear method over the Simplex method for some problems in that it can traverse feasible space. A Simplex method would have taken five more iterations. The execution time was 0.0109 minutes.

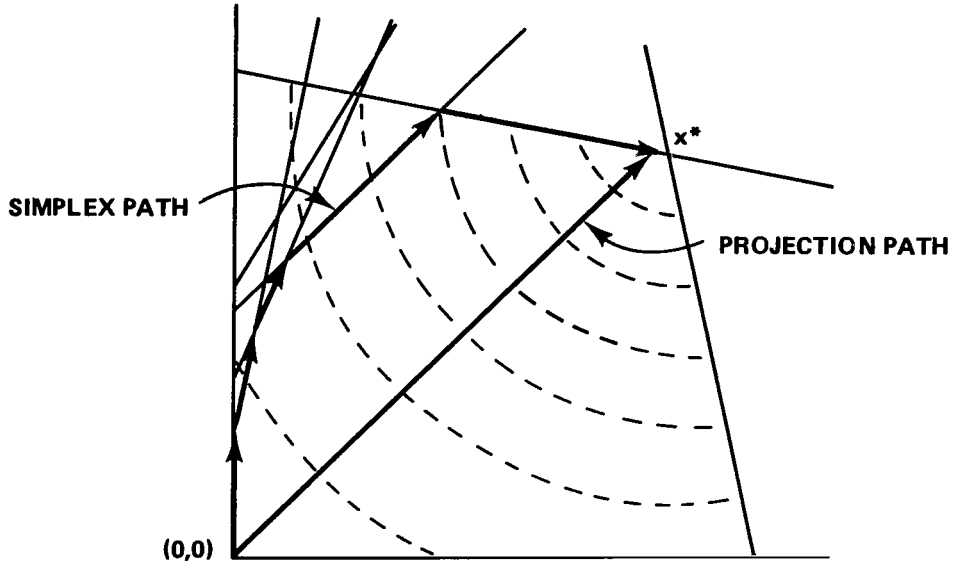


Figure B-3. Advantages of a nonlinear over a linear search.

APPENDIX C. PROGRAM DESCRIPTION

Appendix C contains a brief description of the function of each subroutine of the program and how they interact. The description of each subroutine generally follows the order that data flows through them. In some cases the logic flow in the program is controlled by flags that are passed between subroutines. The program is modular with each of the modules or subroutines generally performing one logical function. Examples of this are found in the MAIN program where the DFP method is located or in subroutine SEEK where the line searches are performed.

MAIN PROGRAM

The main program initializes the parameters, which include the number and type of each constraint, and assesses the supplied initial starting point to determine if it is in feasible space. If it is not or if the problem includes any equality constraints, the MAIN program calls SUBROUTINE FEASPT to internally generate an initial starting point in feasible space. The program then calls SUBROUTINE FUNCT and SUBROUTINE GRAD to get initial values for f(x) and $\nabla f(x)$. The Hessian is initialized to I and SUBROUTINE SEEK is called. For the case where the extremum of the function is located in feasible space or where there are no constraints, then SEEK returns new values for x^i, $f(x^i)$, and $\nabla f(x^i)$ that represent an approximation to the line search extremum. These values are used to calculate an update Hessian matrix. The program tests the new point against the criterion for a local extremum. If it has not satisfied this criterion another search direction is generated and another line search is made. This continues until a local extremum is found or until the maximum iteration limit is acquired. In the case where the extremum is not located in feasible space, the constrained extremum point will be returned to MAIN from SUBROUTINE CHK for printing and program exit.

SUBROUTINE FUNCT and SUBROUTINE GRAD

These are discussed together since they are both user supplied subroutines. The equation for f(x) is supplied by the user in FUNCT, and the gradient functions $\nabla f(x)$ are given in GRAD. The gradient vector returned from GRAD has the proper sign for maximization or minimization.

SUBROUTINE SEEK

This subroutine calculates the DFP search vector when the search is being conducted from the current search point which is unconstrained. When constraints are determined to have been violated, the search vector is supplied by SUBROUTINE CHK. The program is sensitive to the magnitude of the search vector. It adjusts the initial step sizes along this vector inversely to the magnitude of the vector. As it steps along this vector, new values of f(x) are calculated. When a change of direction from decreasing to increasing values of f(x) along the search direction is detected (extremum is bracketed), a quadratic curve fit of the last three points is made in order to estimate the location of the extremum. Figure C-1 shows the curve fit made through $y(\alpha) = f(x^i + \alpha^i H^i \nabla f)$ point for successive α's. This method estimates α^* for the minimum point of a quadratic curve fit where $f(x^{i+1}) = \min[f(x^i + \alpha^i H^i \nabla f(x^i)]$. This approximation of the actual extremum gets better near convergence. In the case where the change of direction in the values of f(x) occurs on the first step along the search vector, the quadratic curve fit is made based on the original starting

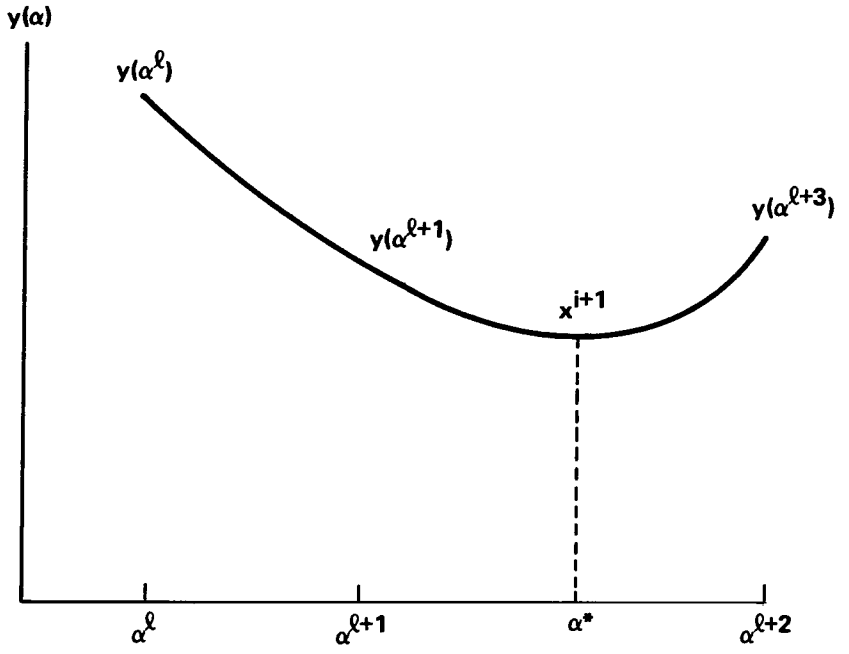


Figure C-1. Extremum estimation along search vector.

point of the search vector, the gradient at that point, and the first step point. SUBROUTINE SEEK takes ten steps along the search vector. If the values of f(x) have not changed direction, then an error flag is returned to the MAIN PROGRAM. If the problem is of such a complexity that a reasonable estimate cannot be made, then SUBROUTINE FIB is called to supply this point. SUBROUTINE CHK is always called to determine if the point found is in feasible space. If it is not, then CHK returns a new search vector. A return is made to MAIN if a constrained extremum was determined.

SUBROUTINE FIB

This subroutine performs an eight-step Fibonacci Search along the vector supplied by SEEK. The new point which is returned is an approximation of the extremum point along the search vector. An eight-step Fibonacci search should be sufficient to provide the proper point within a small error. This type of search method is used since the number of steps have been specified in order to obtain the minimum interval of uncertainty for the placement of eight points.

SUBROUTINE CHK

This subroutine tests for constraint violations. If no violations have occurred, it returns the same search vector to SEEK. When new constraints are violated by a search point, CHK selects those that are violated. These are added to the projection matrix. If the number of these constraints equals or exceeds the number of dimensions, then SUBROUTINE DROP is called to determine which constraints are to be dropped. SUBROUTINE CHK determines when a constrained extremum has been encountered. When either

(1) the projected search vector magnitude becomes smaller than a user set tolerance or (2) the search vector points away from any equality constraint, a constrained extremum is assumed to have been found. When SUBROUTINE DROP returns a new search vector and a new point, SUBROUTINE CHK determines when all new constraints have been acquired and then searches these to see if all of the equality constraints are contained in the projection matrix. If they are not, then the original point sent to DROP is assumed to be the constrained extremum. If the same constraints are acquired again, then this point is assumed to be constrained extremum. The projection matrix is returned from SUBROUTINE SPAN. It uses the set of constraints supplied by CHK.

SUBROUTINE SPAN

This subroutine uses an algorithm that is based on the Gram-Schmidt method to calculate the projection matrix. Any number of new constraints may be added without reinitializing the matrix. When constraints are dropped, all elements in the projection matrix are set to zero and it is recalculated.

SUBROUTINE DROP

In this subroutine the search vector is tested to determine if it points into or out of feasible space. If it points into feasible space, then only the equality constraints are kept to calculate the new projection matrix. If there are no equality constraints, then a flag is set which will cause the program to return to MAIN and initiate the DFP Search method from this point. If the search vector points out of feasible space, then DROP calculates a new starting point at a very small distance from the constraints and sets the search vector equal to the gradient direction. A return is made to CHK to continue the search.

SUBROUTINE FEASPT

This subroutine generates an initial feasible point using Phase I of the two-phase Simplex method. The Phase I method finds a point that lies on the boundary of feasible space which satisfies all of the equality constraints. This point is transferred to the MAIN Program. If Subroutine FEASPT fails to find a feasible point or if more equality constraints are present than the number of variables, a message is printed indicating the type of difficulty encountered, and an error flag is set that terminates the program. When too many equality constraints are present, it indicates that the system has linear dependency, redundancy, or is an inconsistent set.

APPENDIX D. PROGRAM LISTING

```
C*************************************************************
C       THE CONVENTION WILL BE THAT THE NORMALS TO THE
C       CONSTRAINT PLANES WILL BE NEGATIVE FOR THOSE THAT
C       POINT INTO THE FEASIBLE SPACE AND POSITIVE FOR THOSE
C       THAT POINT INTO INFEASIBLE SPACE.
C*************************************************************
C      THE CONSTRAINTS WILL BE ORDERED IN THE PROGRAM IN
C      THE FOLLOWING WAY:
C      NO. OF AX<B CONSTRAINTS = NLEC
C      NO. OF AX=B CONSTRAINTS = NEC
C      NO. OF AX>B CONSTRAINTS = NGEC
C      THE X>O CONSTRAINTS WILL NOT BE IN NGEC
C      NO. OF X>O CONSTRAINTS = NGEX
C*************************************************************
C      N1 = NO. OF DIMENSIONS
C      NN = MAX. NO. OF ITERATIONS
C      EP1 >||X(N+1)-X(N)|| THIS IS FOR CONVERGENCE CRITERIA. EP1=.0001 GENERALLY
C      EP2 >||GRAD(N)|| THIS IS A CONVERGENCE CRITERIA. EP2=.0001 GENERALLY
C      Y = OBJ. FUNCT. VALUE
C      YL = LAST OBJ. FUNCT. VALUE
C      GR1 = GRAD.
C      GRO = LAST GRAD.
C      FLAG1 = 1  SPECIFIES THAT THE 'H' MATRIX HAS BEEN UPDATED N1 TIMES
C                 AND IS COMPLETE. IT WILL BE USED ONE MORE TIME.
C            = 0  SPECIFIES THAT THE 'H' MATRIX IS NOT YET COMPLETE.
C      FLAG2 = 1 USED IN SEEK TO ALERT IT THAT A CONSTRAINT HAS BEEN
C              VIOLATED AND DOES A LINE SEARCH IN PLANE VIOLATED.
C            = 2 CAUSES SEEK TO SET FLAG4 = 1
C      FLAG3 = 1 SPECIFIES THAT THE CONDITIONS FOR A CONSTRAINED
C              MAX(MIN) HAS BEEN MET.
C            = 2 THE MAX. NO. OF TRIES AT FINDING AN EXTREMA ALONG
C                THE SEARCH VECTOR HAS BEEN DONE.
C      FLAG4 = ALERTS PROGRAM THAT IT WAS IN A CONSTRAINT BUT IS
C              NOW IN FEASIBLE SPACE. RESETS H=I,B&C=O AND AAT2=O
C      FLAG5 = ALERTS PROGRAM THAT THERE IS LINEAR DEPENDENCE AMONG
```

```
C              THE EQUALITY CONSTRAINTS.
C      KK = THIS IS A COUNT OF THE TOTAL NO. OF ITERATIONS THROUGH THE
C           HESSIAN. IT IS USED TO FORCE THE PROG. THROUGH AT LEAST
C           N1 UPDATES OF 'H'. USED TO ABORT PROG. IF ERROR BOUNDS
C           CAN'T BE MET BY SPECIFIED NO. OF ITERATIONS.
C      K = RESETS d = I AFTER N1 ITERATIONS.
C      NC = NO. OF CONSTRAINTS INCLUDING THE Xi>0 CONSTRAINTS
C      NUM = THE VARIABLE THAT DESIGNATES EITHER MAX OR MIN.
C            IT DOES THIS BY IT'S SIGN (+/ IS MAX, -/ IS MIN)
       COMMON XO(10),X1(10),H(10,10),A(20,10),GRO(10),BC(20),
      1N1,NC,AAT2(10,10),KOF,NCVO(20),NLEC,NEC,NGEC,NGEX,KC
       DIMENSION GR1(10),XD(10),GRD(10),P(10),
      1B(10,10),B1(10),C(10,10),CH(10),CHN(10,10)
       INTEGER FLAG1,FLAG2,FLAG3,FLAG4,FLAG5,NC,TYPE,QMIN,QMAX
       DATA QMAX,QMIN/4HMAX ,4HMIN /
C***** KOF=0 MEANS THAT THE SEARCH MUST START IN FEASIBLE SPACE.
C      INITIALIZE
       FLAG5=0
       KOF=0
       KK=0
       KC=0
       K=0
       NUM=1
       READ(5,300)TYPE,N1,NN,NC,EP1,EP2,NLEC,NEC,NGEC,NGEX
       WRITE(6,320)TYPE,N1,NN,NC,EP1,EP2,NLEC,NEC,NGEC
  320 FORMAT('WE ARE LOOKING FOR A ',A3/
     1'NO. OF DIMENSIONS =',I3/'MAX. NO. OF ITERATIONS =',I3/
     2'NO. OF CONSTRAINTS =',I3/'EP1 & EP2 =',2E13.6/
     3'NO. OF LE CONSTRAINTS=',I3/'NO.OF = CONSTRAINTS=',I3/
     4'NO. OF GE CONSTRAINTS=',I3)
       READ(5,301)(XO(I),I=1,N1)
       WRITE(6,321)N1,(XO(I),I=1,N1)
  321 FORMAT('XO(I),I=1,N1'/NE13.6)
       IF(NC.EQ.0)GO TO 33
       DO 34 I=1,NC
```

```
   34 READ(5,301)BC(I)
      DO 1 I=1,NC
    1 READ(5,301)A(I,J),J=1,N1
      WRITE(6,322)
  322 FORMAT('THIS IS THE "A" MATRIX')
      NO=NC-NGEX
      WRITE(6,318)(N1,(A(I,J),J=1,N1),I=1,NO)
  318 FORMAT(NE13.6)
      WRITE(6,400)NO,(BC(I),I=1,NO)
  400 FORMAT('BC= ',NE13.6)
C***** IF THERE ARE EQUALITY CONSTRAINTS THEN THE PHASE 1 METHOD
C      WILL BE USED AS THIS WILL PUT THE INITIAL POINT ON THE
C      EQUALITY CONSTRAINTS AS REQUIRED.
      IF(NEC.NE.0)GO TO 32
      DO 29 I=1,NC
      BC1=0.0
      DO 28 J=1,N1
   28 BC1=A(I,J)*X0(J)+BC1
      IF(I.GT.NLEC)GO TO 40
      IF((BC(I)-BC1).LT.0.0)GO TO 32
      GO TO 29
   40 IF(I.GT.(NLEC+NEC))GO TO 41
      IF(ABS(BC(I)-BC1).GT.1.E-12)GO TO 32
      GO TO 29
   41 IF((BC(I)-BC1).GT.0.0)GO TO 32
   29 CONTINUE
      GO TO 33
   32 CONTINUE
C***** THIS ROUTINE IS THE SIMPLEX PHASE 1 METHOD. IT PIVOTS
C      THROUGH THE VERTICIES UNTIL IT LOCATES ONE IN FEASIBLE SPACE.
      CALL FEASPT(FLAG5)
      IF(FLAG5.EQ.1)GO TO 36
C***** IT IS ASSUMED THAT THE CONSTRAINTS ARE LINEARLY INDEPENDENT.
      IF(N1.GT.NEC)GO TO 33
      DO 35 I=1,N1
```

```
   35 X1(I)=X0(I)
      GO TO 23
   33 CONTINUE
      IF(QMAX.EQ.TYPE)NUM=1
      IF(QMIN.EQ.TYPE)NUM=-1
  323 FORMAT('THE DATA HAS BEEN READ')
C*****X0 = INITIAL POINT
C*****Y = VALUE OF THE OBJ. FUNCT.
C*****N1 = NO. OF DIMENSIONS
C*****NC = NO. OF CONSTRAINTS
      CALL FUNCT(X0,Y,N1)
  210 CONTINUE
      FLAG3=0
      YL=Y
  350 FORMAT('X0(I),Y'/NE14.7)
C     GRAD. OF THE FUNCT. AT THE POINT X0
      CALL GRAD(X0,GR0,N1,NUM)
  351 FORMAT('GR0(I)'/NE14.7)
  100 CONTINUE
C*****THIS SETS THE HESSIAN = I
      DO 3 I=1,N1
      DO 2 J=1,N1
      AAT2(I,J)=0.0
      H(I,J)=0.0
    2 IF(I.EQ.J)H(I,J)=1.0
      NCV0(I)=0
    3 CONTINUE
      FLAG1=0
      FLAG4=0
      K=0
      YL=Y
  309 CONTINUE
C
C*****THIS IS THE LINE SEARCH MAX(MIN)f(X1+@HGRAD)
C
```

```
      CALL SEEK(Y,YL,FLAG1,FLAG3,FLAG4,NUM,RHO,P)
C
C***** IF FLAG3=2 THE MAX NO. OF ATTEMPTS HAS BEEN DONE ALONG
C***** THE SEARCH VECTOR.
C
      IF(FLAG3.EQ.2)GO TO 27
C
C***** IF FLAG3=1 A CONSTRAINED MAX(MIN) HAS BEEN FOUND
C
      IF(FLAG3.EQ.1)GO     TO 23
C
C***** IF FLAG4=1 THE GRAD NOW POINTS BACK INTO FEASIBLE SPACE
C
      IF(FLAG4.EQ.1)GO     TO 200
C
C***** IF FLAG1=1 A LINE SEARCH WITH THE COMPLETE HESS HAS BEEN
C      COMPLETED AND NOW RESET HESS=I
C
      IF(FLAG1.EQ.1)GO     TO 22
      CALL GRAD(X1,GR1,N1,NUM)
      WRITE(6,314)
  314 FORMAT(5X,'X1',12X,'X0',12X,'GR1',12X,'GR0')
      WRITE(6,315)(X1(I),X0(I),GR1(I),GR0(I)),I=1,N1
  315 FORMAT(4E14.7)
      DO 4 I=1,N1
      XD(I)=X1(I)-X0(I)
    4 GRD(I)=(GR1(I)-GR0(I))*NUM
      B2=0.0
      C2=0.0
      DO 20 I=1,N1
      B2=B2+XD(I)**2
   20 C2=C2+GRD(I)**2
      B2=SQRT(B2)
      C2=SQRT(C2)
      GO TO 22
```

```
  200 CONTINUE
      IF(FLAG1.EQ.1)GO      TO 19
C*****RESETS THE B&C MATRICES = 0
      KOF=0
      DO 16 I=1,N1
      NCVO(I)=0
      CH(I)=0.0
      DO 15 J=1,N1
      AAT2(I,J)=0.0
      B(I,J)=0.0
   15 C(I,J)=0.0
   16 CONTINUE
      IF(FLAG4.EQ.1)GO      TO 100
      B3=0.0
      DO 5 I=1,N1
    5 B3=B3+P(I)*GRD(I)
      B3=ABS(RHO)/B3
C 306 FORMAT('B3=',E14.7)
      DO 6 I=1,N1
      DO 6 J=I,N1
      B(I,J)=P(I)*P(J)*B3
    6 B(J,I)=B(I,J)
      DO 7 I=1,N1
      CH(I)=0.
      DO 7 J=1,N1
    7 CH(I)=CH(I)+H(I,J)*GRD(J)
      CHD=0.
      DO 8 I=1,N1
    8 CHD=CHD+GRD(I)*CH(I)
      DO 9 I=1,N1
      DO 9 J=1,N1
    9 CHN(I,J)=CH(I)*GRD(J)
      DO 11 I=1,N1
      DO 11 J=1,N1
      C(I,J)=0.
```

```
      DO 10 J1=1,N1
   10 C(I,J)=C(I,J)+CHN(I,J1)*H(J1,J)
   11 C(I,J)=C(I,J)/CHD
      DO 12 I=1,N1
      DO 12 J=1,N1
   12 H(I,J)=H(I,J)+B(I,J)-C(I,J)
   19 CONTINUE
      KK=KK+1
      K=K+1
      DO 21 I=1,N1
      XO(I)=X1(I)
   21 GRO(I)=GR1(I)
      YL=Y
      GO TO 309
  250 CONTINUE
      DO 251 I=1,N1
      XO(I)=X1(I)
  251 CONTINUE
      GO TO 210
   22 CONTINUE
C
C*****MUST GO THROUGH AT LEAST N1 CALC'S OF 'H' FIRST TIME
C
      IF(KK.LT.N1)GO     TO 24
C
C*****THIS CHECK IS TO DETERMINE IF YOU'RE CLOSE ENOUGH
C
      IF(B2.LT.EP1.AND.C2.LT.EP2)GO TO 23
   24 CONTINUE
      IF(FLAG1.EQ.1)GO     TO 100
      IF(K.EQ.N1)FLAG1=1
C
C*****IF KK=MAX NO. OF ITERATIONS THEN STOP
C
      IF(KK.EQ.NN)GO TO 23
```

```
C
C*****THIS ALLOWS ONE STEP WITH THE COMPLETED 'H' MATRIX
C
      GO TO 200
   23 CONTINUE
      IF(FLAG3.EQ.0)GO TO 27
      WRITE(6,305)TYPE
  305 FORMAT('A CONSTRAINED ',A3,' HAS BEEN FOUND.')
   27 CONTINUE
      IF(FLAG3.EQ.0)GO TO 37
      IF(FLAG3.NE.2)GO TO 13
      WRITE(6,324)
  324 FORMAT('THE LINE SEARCH HAS TAKEN THE MAX. NO. OF STEPS')
      CALL GRAD(X1,GR1,N1,NUM)
      GO TO 37
   13 CONTINUE
      DO 38 I=1,N1
      X1(I)=X0(I)
   38 GR1(I)=GR0(I)
   37 CONTINUE
      WRITE(6,303)KK,KC
  303 FORMAT('NO. OF UNCONSTRAINED ITERATIONS=',I4//
     *'NO. OF CONSTRAINED ITERATIONS=',I4)
      WRITE(6,304)Y,N1,(X1(I),I=1,N1),N1,(GR1(I),I=1,N1)
  304 FORMAT('F(X*)=',E14.7//'X*=',NE13.6//
     *'GRADIENT F(X*)=',NE13.6)
  300 FORMAT(A3,3I3,2E14.7,4T3)
  301 FORMAT(10E13.6)
   36 CONTINUE
      END
```

```
      SUBROUTINE FUNCT(A,B,N)
      DIMENSION A(N)
      B=100.*(A(2)-A(1)**2)**2+(1.-A(1))**2+
     *100.*(A(3)-A(2)**2)**2+(1.-A(2))**2
      RETURN
      END



      SUBROUTINE GRAD(A,B,N,NUM)
      DIMENSION A(N),B(N)
      B(1)=-40.*(A(2)-A(1)**2)*A(1)-2.*(1.-A(1))
      B(2)=20.*(A(2)-A(1)**2)-2.*(1.-A(2))-40.*(A(3)-A(2)**2)*A(2)
      B(3)=20.*(A(3)-A(2)**2)
      IF(NUM.GT.0)RETURN
      DO 1 I=1,N
    1 B(I)=NUM*B(I)
      RETURN
      END
```

```
      SUBROUTINE SEEK(Y,YL,FLAG1,FLAG3,FLAG4,NUM,RHOM,DD)
C     B = B MATRIX OF CONSTRAINTS(A'X=B)
C     NC = NO. OF CONSTRAINTS
C     A = MATRIX OF THE CONSTRAINTS COEFFICIENTS
C
C     FLAG1 = 1 ALERTS MAIN PROG. THAT THERE IS A NUMERICAL
C               PROBLEM AND TO RESET H=I
C
C     FLAG2 = 1 ALERTS SEEK THAT CHK IS RETURNING A NEW
C               SEARCH VECTOR. IT IS RESET IN CHK WHEN NO
C               NEW CONSTRAINTS ARE VIOLATED OR THE SEARCH
C               VECTOR POINTS INTO FEAS. SPACE WITH NO
C               EQUALITY CONSTRAINTS PRESENT.
C
C     FLAG2 = 2 CAUSES FLAG4=1 AND RETURNS TO THE MAIN PROG.
C               TO DO THE DFP METHOD.
C
C     FLAG3 = 1 ALERTS PROG. THAT A BOUND EXTREMA IS LOCATED.
C
C     FLAG3 = 2 ALERTS PROG. THAT THE LINE SEARCH CANNOT FIND A
C               BETTER VALUE THAN THE LAST ONE. (MAX. STEPS)
C     FLAG4 = 1 ALERTS MAIN PROG. TO INITIATE THE DFP METHOD.
C
C               FOR SUBROUTINE CHK.
C     FLAG7 = 0 ALERTS CHK THAT THREE STEPS ALONG THE LINE
C               SEARCH WERE MADE( MAX NO.) WITHOUT FINDING AN EXTREMA.
C
C     FLAG7 = 1 ALERTS CHK THAT AN EXTREMA WAS LOCATED ALONG
C               THE SEARCH VECTOR.
C
C     FLAG8 = 1 SET WHEN GOING FROM TWO POINT FIT TO A THREE
C               POINT METHOD. SET = 0 OTHERWISE.
C
      COMMON X0(10),X1(10),H(10,10),A(20,10),GR0(10),BC(20),
     1N1,NC,AAT2(10,10),KOF,NCVO(20),NLEC,NEC,NGEC,NGEX,KC
```

```
      DIMENSION DD(10)
      INTEGER FLAG1,FLAG2,FLAG3,FLAG4,FLAG6,FLAG7,FLAG8
      DATA ETOL1,ETOL2/10.,1.E-5/
      IO=0
      ICNT=0
      FLAG2=0
      FLAG6=0
      FLAG7=0
      DO 205 I=1,N1
  205 DD(I)=0.0
C
C*****DD=H*GRAD IS THE SEARCH VECTOR.
C
      DO 150 I=1,N1
      DO 149 J=1,N1
  149 DD(I)=H(I,J)*GRO(J)+DD(I)
  150 CONTINUE
  151 CONTINUE
      FLAG8=0
      WRITE(6,307)N1,(DD(I),I=1,N1)
  307 FORMAT(4X,'SEARCH VECTOR IN LINE SEARCH'/NE13.6)
C
C***** DETECTION OF THE CONDITION FOR AN UNBOUNDED SOLUTION WILL BE
C      DONE. IF THESE CONDITION ARE MET THEN A MESSAGE WILL BE PASSED ALONG BUT
C      THE PROG. WILL NOT BE STOPPED( COULD STILL FIND AN EXTREME VALUE).
C      ALL CONSTRAINTS IN WHICH THE DOT PRODUCT OF THE SEARCH VECTOR
C      AND NORMAL TO THE CONSTRAINT PLANE ARE >/=0 ARE NOT BINDING.
      IDR=-1
      DO 12 I=1,NC
      IF(I.GT.NLEC)IDR=1
      UNBND=0.0
      DO 12 J=1,N1
      UNBND=UNBND+DD(J)*IDR*A(I,J)
C
C***** AT LEAST ONE CONSTRAINT WITH A NEG. DOT PRODUCT DENOTES A BINDING
```

```
C       CONSTRAINT HAS BEEN FOUND.
C
        IF(UNBND.LT.0.0)GO TO 13
        IF(I.NE.NC)GO TO 12
        WRITE(6,310)
  310 FORMAT('CONDITIONS FOR AN UNBOUNDED SOLUTION HAS BEEN DETECTED')
        GO TO 13
   12 CONTINUE
   13 CONTINUE
        DMP=0.0
        DM=0.0
C
C*****DMP IS USED TO SET THE SENSITIVITY OF THE LINE SEARCH TO THE
C       MAGNITUDE OF THE HGRAD.
C
        DO 3 I=1,N1
    3 DMP=DMP+DD(I)*DD(I)
        DMP=SQRT(DMP)
        PT=DMP
        DO 18 I=1,N1
   18 DM=DM+NUM*GRO(I)*DD(I)/DMP
C
C***** DM = THE GRAD. OF A FUNCT  OF  RHO. (XO+RHO*HGRAD)
C
        IF(DMP.GT.10000.)RHO=.001
        IF(DMP.GE.200..AND.DMP.LE.10000.)RHO=10./DMP
        IF(DMP.GT.5..AND.DMP.LT.200)RHO=.1
        IF(DMP.LE.5)RHO=1.
        WRITE(6,300)
  300 FORMAT('THIS IS H(I,J) IN LINE SEARCH')
        WRITE(6,303)(N1,(H(I,J),J=1,N1),I=1,N1)
  303 FORMAT(NE13.6)
C
C*****IF THE HESS IS BADLY CONDITIONED OR HAS EXCESSIVE ROUNDOFF ERROR
C       THE SEARCH VECTOR MAY HAVE THE WRONG SIGN.
```

```
C      HENCE DD*GRAD=+ ALWAYS UNLESS SOMETHING IS WRONG.
C***** THIS TEST ON THE CONDITION OF SEARCHING IN THE CONSTRAINT MANIFOLD.
C
       IF(FLAG2.EQ.1)GO      TO 10
       IF(DM*NUM.LT.0.0)GOTO 7
C
C***** STEP DOWN THE SEARCH VECTOR FOR AT LEAST THREE STEPS
C       THE LATER INTERPOLATION USES THREE POINTS
C
    10 CONTINUE
       LL=0
       RHO1=RHO
       RHOO=0.0
       RHOOO=0.0
       YO=YL
       YOO=YL
   110 CONTINUE
       DO 120 I=1,N1
   120 X1(I)=XO(I)+DD(I)*RHO
       CALL FUNCT(X1,Y1,N1)
C
C*****THE NEXT IF HAS (Y1.GE.YO) WHEN MIN. AND (Y1.LE.YO) FOR MAX.
C
       IF(NUM.LT.0..AND.Y1.GE.YO)GO TO 2
       IF(NUM.GT.0..AND.Y1.LE.YO)GO TO 2
       IF(LL.EQ.0)GO TO 1
       YOO=YO
       RHOOO=RHOO
     1 YO=Y1
       RHOO=RHO
       LL=LL+1
C
C*****IF MAX(MIN) HASN'T BEEN DETECTED YET THEN CALC. A PT. ANYWAY.
C
       IF(LL.EQ.3.AND.FLAG2.EQ.1)GO      TO 8
```

```
      IF(LL.LE.10)GO TO 22
      FLAG3=2
      Y=Y1
      GO TO 8
   22 CONTINUE
      RHO=RHO1*2.**LL
      RHOM=RHO
      GO TO 110
C
C*****THIS CATCHES THE CASE WERE THE FIRST PT. STRADDLES THE EXTREMA.
C
    2 IF(LL.EQ.0)GO TO 6
C
C
C***** THE ABOVE CATCHES CASE TWO (y'(d1),y(d1),y(d2),d1,d2 ARE KNOWN)
C***** CASE ONE IS WHERE y(d1),y(d2),y(d3),d1,d2,d3 ARE KNOWN.
C
C
   19 CONTINUE
      XX=(RHOO-RHOOO)*(Y1-YOO)-(RHO-RHOOO)*(YO-YOO)
      RHOM=RHOOO+.5*(((RHOO-RHOOO)**2*(Y1-YOO)-(RHO-RHOOO)**2*
     X(YO-YOO))/XX)
      IF(ABS(XX).LT..0000000001)RHOM=.000001
    4 CONTINUE
      DO 5 I=1,N1
    5 X1(I)=XO(I)+DD(I)*ABS(RHOM)
      WRITE(6,308)
  308 FORMAT(4X,'X1(I)',9X,'XO(I)',9X,'DD(I)')
      WRITE(6,306)(X1(I),XO(I),DD(I)),I=1,N1
  306 FORMAT(3E13.6)
      CALL FUNCT(X1,Y,N1)
C
C***** FLAG7=1 ALERTS CHK THAT AN EXTREMA ON THE LINE SEARCH
C              WAS FOUND
C
```

```
      FLAG7=1
      IF(NUM.LT.0.AND.Y.LE.YO)GO TO 8
      IF(NUM.GT.0.AND.Y.GE.YO)GO TO 8
C
C***** GIVE IT TEN TRIES AND THEN CHECK TO SEE IF ITS CLOSE ENOUGH.
C
      ICNT=ICNT+1
      IF(ICNT.GT.10)GO TO 15
      IF(FLAG8.EQ.1)GO TO 17
      IF(LL.NE.0)GO TO 14
      RHOO=RHOM
      YO=Y
      FLAG8=1
      GO TO 19
   14 IF(ABS(RHOM).GT.RHOO)GO TO 20
      RHOOO=RHOM
      YOO=Y
      GO TO 19
   20 CONTINUE
      RHO=RHOM
      Y1=Y
      GO TO 19
   17 RHO=RHOO
      Y1=YO
      RHOO=RHOM
      YO=Y
      GO TO 19
   15 CONTINUE
      IF(ABS(ABS(Y)-ABS(YO)).LT.1.E-5*ABS(YO))GO TO 8
   16 CONTINUE
      FLAG3=2
      WRITE(6,312)
  312 FORMAT('SEARCH FAILED TO FIND A BETTER VALUE THAN XO')
      CALL GRAD(XO,GRO,N1,NUM)
      GO TO 9
```

```
C
C***** THIS IS THE FINAL ADJUSTMENT OF THE SEARCH STEP SIZE
C
    6 IF(ABS(Y1).LT.(100.*ABS(YO)))GO TO 21
      RHO=.1*RHO
      GO TO 10
   21 CONTINUE
      RHOM=.5*RHO**2*DM/(RHO*DM+YO-Y1)
      IF(ABS(RHO*DM+YO-Y1).LT..0000000001)RHOM=.000001
C
C***** IF THE GRAD OR PROJECTION ARE LARGE AND THE SEARCH STEP SIZE IS
C      SMALL THEN MORE ACCURACY IS CALLED FOR THAN THE QUADRATIC FIT
C      CAN GIVE SO SUBROUTINE FIB IS CALLED.
C
      IF(PT.GT.ETOL1.AND.RHOM.LT.ETOL2)CALL FIB(RHO,RHOM,NUM,XO,N1,DD)
      GO TO 4
    7 FLAG1=1
    8 CONTINUE
      IF(NC.EQ.0)GO TO 9
C
C***** THIS ROUTINE CHECKS TO SEE IF ANY CONSTRAINTS ARE VIOLATED
C      AND IF SO IT RETURNS WITH A NEW SEARCH VECTOR
C
      CALL CHK(PT,DD,FLAG2,FLAG3,FLAG6,NUM,FLAG7,IO)
C
C***** FLAG3=1 A CONSTRAINED MAX(MIN) HAS BEEN FOUND
C
      IF(FLAG3.EQ.1)GO     TO 9
C
C***** FLAG2=1 CONSTRAINT(S) HAVE BEEN VIOLATED AND CHK HAS
C      SUPPLIED A NEW SEARCH VECTOR.
C
      FLAG7=0
      IF(FLAG2.NE.1)GO     TO 11
      CALL FUNCT(XO,YL,N1)
```

```
      ICNT=0
      GO TO 151
   11 IF(FLAG2.EQ.2)FLAG4=1
    9 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE FIB(RHO,RHOM,NUM,X0,N1,DD)
C
C***** THIS IS AN 8 STEP FIBONACCI SEARCH
C
      DIMENSION X0(N1),X1(10),XT1(10),DD(N1)
      DATA ETOL/1.E-5/
      REAL L2
      ISTEP=1
      RANU=RHO
      RANL=0.
      L2=(RHO*13.+(ETOL*(-1)**8))/21.
      X2=RHO-L2
    1 CONTINUE
      IF(ISTEP.EQ.8)GO TO 13
      ISTEP=ISTEP+1
      DO 2 I=1,N1
      XT1(I)=X0(I)+DD(I)*L2
    2 X1(I)=X0(I)+DD(I)*X2
      CALL FUNCT(XT1,Y1,N1)
      CALL FUNCT(X1,Y2,N1)
      IF(Y1.LT.Y2.AND.X2.GT.((RANU-RANL)/2.+RANL))GO TO 10
      IF(Y1.LT.Y2.AND.X2.LT.((RANU-RANL)/2.+RANL))GO TO 7
      IF(Y1.GT.Y2.AND.X2.GT.((RANU-RANL)/2.+RANL))GO TO 6
      IF(NUM.GT.0)GO TO 12
    3 RANU=L2
    4 CONTINUE
      IF(X2.GT.((RANU-RANL)/2.+RANL))GO TO 5
      L2=(RANU-X2)+RANL
      GO TO 1
    5 L2=RANU-(X2-RANL)
      GO TO 1
    6 CONTINUE
      IF(NUM.GT.0)GO TO 11
   14 RANL=L2
      GO TO 4
```

```
 7 CONTINUE
   IF(NUM.GT.0)GO TO 3
12 RANL=X2
 8 CONTINUE
   IF(L2.GT.((RANU-RANL)/2.+RANL))GO TO 9
   X2=(RANU-L2)+RANL
   GO TO 1
 9 X2=RANU-(L2-RANL)
   GO TO 1
10 CONTINUE
   IF(NUM.GT.0)GO TO 14
11 RANU=X2
   GO TO 8
13 CONTINUE
   RHOM=(RANU-RANL)/2.+RANL
   RETURN
   END
```

```
      SUBROUTINE CHK(PT,P,FLAG2,FLAG3,FLAG6,NUM,FLAG7,IO)
C*****NCV - THE NUMBER OF THE VIOLATED CONSTRAINT
C     NCS - THE NUMBER OF THE VIOLATED CONSTRAINTS (NCV(I))
C           THAT HAVE THE LOWEST ALPHA. THIS IS USED TO
C           INDEX NCV.
C***** EX:
C         NCV(NCS(2)) - IS THE SECOND OF THE NEWLY VIOLATED
C         CONSTRAINTS THAT HAVE ALPHA'S EQUAL TO THE LOWEST
C         ALPHA (MINIMUM SET VIOLATED).
C
C     FLAG2 = 0 ALERTS CHK THAT THE PREVIOUS  CYCLE DIDN'T
C               VIOLATE A CONSTRAINT.
C
C     FLAG2 = 1 ALERTS CHK THAT CONSTRAINTS HAVE ALREADY BEEN VIOLATED.
C
C     FLAG3 = 1 ALERTS PROG. THAT A BOUND EXTREMA HAS BEEN FOUND.
C
C     FLAG6 > 0 ALERTS CHK THAT DROP HAS GENERATED A NEW POINT.
C
C     FLAG6 = 2 ALERTS CHK TO LOOK FOR THE FURTHEREST CONSTRAINT
C               FROM POINT RETURNED BY DROP AS IT IS IN
C               INFEASIBLE SPACE (SEARCH VECTOR POINTS IN).
C
C     FLAG7 = 1 ALERTS CHK THAT AN EXTREMA WAS FOUND ON LINE SEARCH.
C
      COMMON XO(10),X1(10),H(10,10),A(20,10),GRO(10),BC(20),
     1N1,NC,AAT2(10,10),KOF,NCVO(20),NLEC,NEC,NGEC,NGEX,KC
      DIMENSION B(20),NCV(20),SAX(20),SAND(20),ALPHA(20),
     1NCS(20),XN(10),P(10),GRN(10),HGRO(20),NTOC(20),
     2MNB(20),NCV1(20),GRT(10),XT(10)
      INTEGER FLAG1,FLAG2,FLAG3,FLAG4,FLAG6,FLAG7
      REAL IDENT
      DATA ETOL/1.E-5/
      WRITE(6,313)
  313 FORMAT('THE X1 POINT')
```

```
      WRITE(6,314)N1,(X1(I),I=1,N1)
  314 FORMAT(NE13.6)
      KF=0
      FLAG4=0
      BT=0.0
      SAX(0)=0.0
      SAHD(0)=0.0
      DO 14 I=1,NC
      B(I)=0.0
      HGRO(I)=0.0
   14 CONTINUE
C
C***** THIS CALCULATES THE ACTUAL VALUES OF THE CONSTRAINT EQ.'S
C
      DO 1 I=1,NC
      DO 1 J=1,N1
      B(I)=A(I,J)*X1(J)+B(I)
      IF(I.GT.N1)GO     TO 1
      IF(FLAG2.EQ.1)GO     TO 1
C
C*****ONLY NEED HGRO WHEN GOING FROM FEASIBLE SPACE TO CONSTRAINT.
C
      HGRO(I)=H(I,J)*GRO(J)+HGRO(I)
    1 CONTINUE
C*****************************************************************
C*****************************************************************
C     SORT THE CONSTRAINTS FOR THOSE THAT ARE VIOLATED.
C       KA      =NO. OF NEWLY VIOLATED CONSTRAINTS
C       KO      = "   " OLD CONSTRAINTS STILL VIOLATED
C       KOF     = "   "   "  CONSTRAINT VIOLATIONS
C       NCVO(I)= "   " THE OLD VIOLATED CONSTRAINT
C       NCV1(I)= "   "   "    "      "          " STILL VIOLATED
C       NCV(I) = "   "   "  NEWLY VIOLATED CONSTRAINT
C*****************************************************************
C*****************************************************************
```

```
      KA=0
      KO=0
      DO 2 I=1,NC
      BF=BC(I)-B(I)
C
C***** THIS IS HERE TO ALLEVIATE NUMERICAL DIFFICULTIES
C
      IF(ABS(BF).LT.ETOL.AND.FLAG2.EQ.1)GO  TO 51
      IF(FLAG6.NE.2)GO TO 62
C**********************************************************************
C**********************************************************************
C       THE FOLLOWING SECTION IS HERE TO CHEK FOR A NEW CONSTRAINT
C       BEING ENCOUNTERED WHEN THE SEARCH VECTOR POINTS INTO
C       FEASIBLE SPACE AFTER THE PROGRAM HAS STEPPED INTO
C       INFEASIBLE SPACE.
C**********************************************************************
C**********************************************************************
      IF(BF.LT.0..AND.I.LE.NLEC)GO TO 2
      IF(BF.GT.0..AND.I.GT.NLEC+NEC)GO TO 2
      DO 64 II=1,KOFO
      WRITE(6,328)I,NTOC(II)
  328 FORMAT('I=',I3,' NTOC=',I3)
   64 IF(I.EQ.NTOC(II))GO TO 46
      GO TO 2
   62 CONTINUE
      IF(BF.GE.0.0.AND.I.LE.NLEC)GO TO 2
      IF(BF.LE.0.0.AND.I.GT.(NLEC+NEC))GO TO 2
   51 CONTINUE
      IF(KOF.EQ.0)GO TO 46
      DO 45 K=1,KOF
   45 IF(NCVO(K).EQ.I)GO TO 50
   46 CONTINUE
      KA=KA+1
      NCV(KA)=I
      GO TO 2
```

```
   50 KO=KO+1
      NCV1(KO)=I
    2 CONTINUE
C
C***** THIS TESTS TO SEE IF ANY NEW CONSTRAINTS WERE VIOLATED
C       AFTER THE BRANCH IT TESTS TO SEE IF IT WAS ALREADY
C       IN A CONSTRAINT PLANE(SEE COMMENT BELOW).
C
      IF(KA.EQ.0)GO TO 13
      WRITE(6,300)KA,(NCV(I),I=1,KA)
  300 FORMAT(I3,' CONSTRAINTS HAVE BEEN VIOLATED. THEY ARE',20I4)
C*************************************************************
C*************************************************************
C***** FLAG4 IS SET WHEN THE NEXT XO BEING LOOKED FOR IS DOWN A
C       SEARCH VECTOR ALREADY IN A PLANE THAT HAS VIOLATED AT
C       LEAST ONE OTHER PLANE.
C***** FROM HERE TO STATEMENT 4 A SIMULTANEOUS SOLUTION OF
C       THE SEARCH VECTOR AND THE CONSTRAINT PLANES IS CALCULATED
C       THIS ACCOMPLISHES TWO THINGS
C       (1)     AN EXACT PT. (X1) IN THE PLANE IS LOCATED
C       (2) AN EXACT DISTANCE(ALPHA) FROM XO TO X1 IS CALCULATED
C*************************************************************
C*************************************************************
      FLAG4=FLAG2
      FLAG2=1
      FLAG3=0
      DO 15 I=1,KA
      SAX(I)=0.0
      SAHD(I)=0.0
   15 CONTINUE
      DO 4 I=1,KA
      DO 3 K=1,N1
      SAX(I)=SAX(I)+A(NCV(I),K)*XO(K)
      IF(FLAG4.EQ.0)GO TO 24
      SAHD(I)=SAHD(I)+NUM*A(NCV(I),K)*P(K)
```

```
      GO TO 3
   24 CONTINUE
      SAHD(I)=SAHD(I)+NUM*A(NCV(I),K)*HGRO(K)
    3 CONTINUE
    4 ALPHA(I)=(BC(NCV(I))-SAX(I))/SAHD(I)
C
C*****THIS ALPHA IS IN THE EQ. X1=XO+ALPHA*HGRO .IT WILL BE USED
C     TO ASCERTAIN THE SMALLEST BORDER OF THE FEASIBLE SPACE
C     THIS WILL BE DONE BY SORTING FOR THE SMALLEST ALPHA
C
C     DO 16 I=1,KA
C     DO 16 K=1,N1
C
C*****SORT FOR THE LOWEST ALPHA OF THE NEWLY VIOLATED CONSTRAINTS
C***** IF FLAG6 = 2 FIND THE LARGEST ALPHA( SEARCH FROM INFEAS. SPACE)
C
      TRIAL=1.E-7
      IF(FLAG6.GT.0)TRIAL=1.E-14
      NM=1
      NCS(1)=1
      ALPO=ABS(ALPHA(1))
      DO 5 I=1,KA
      IF(FLAG6.GT.0)GO TO 38
      IF(ABS(ABS(ALPHA(I))-ALPO).LT.TRIAL)GO        TO 52
   38 CONTINUE
      IF(FLAG6.NE.2)GO TO 43
      IF(ABS(ALPHA(I)).LT.ALPO)GO TO 5
      GO TO 52
   43 CONTINUE
      IF(ABS(ALPHA(I)).GT.ALPO)GO TO 5
   52 IF(I.EQ.1)GO     TO 5
      ALPO=ABS(ALPHA(I))
      NM=I
    5 CONTINUE
      K=1
```

```
      IF(FLAG6.EQ.2)FLAG6=0
      NFLAG10=0
C
C*****ONLY ONE CONSTRAINT HAS THE LOWEST ALPHA.
C
      IF(NM.EQ.1)GO     TO 7
C
C*****HOW MANY OF THESE CONSTRAINTS HAVE ALPHA'S = TO THE LOWEST
C     ALPHA AND WHA' IS THEIR NUMBER( SET NCS).
C
      DO 6 I=1,NM
      IF(ABS(ABS(ALPHA(I))-ALPO).GT.TRIAL)GO        TO 6
      NCS(K)=I
      K=K+1
    6 CONTINUE
      IF(NM.EQ.2)NCS(2)=2
      K=K-1
    7 CONTINUE
      KF=K
C*********************************************************************
C*********************************************************************
C
C      THIS SECTION EXAMINES THE NEWLY ACQUIRED CONSTRAINTS.
C      IF IT DETERMINES THAT IT HAS FOUND A NEW ONE THEN
C      FLAG6 IS RESET TO 0. FLAG6 IS NOT ZERO IF IT HAS BEEN
C      THROUGH DROP. NTOC CONTAINS ALL OF THE PREVIOUSLY
C      VIOLATED CONSTRAINTS.
C
C*********************************************************************
C*********************************************************************
      IF(FLAG6.EQ.0)GO TO 63
      DO 68 I=1,KF
      DO 66 II=1,KOFO
   66 IF(NCV(NCS(I)).EQ.NTOC(II))GO TO 68
      WRITE(6,335)NCV(NCS(I))
```

```
  335 FORMAT(I3,' DOES NOT MATCH THOSE THAT WERE DROPPED')
      WRITE(6,381)KOFO,(NTOC(II),II=1,KOFO)
      FLAG6=0
      GO TO 63
   68 CONTINUE
      IF(KO.EQ.0)GO TO 73
      DO 69 I=1,KO
      DO 67 II=1,KOFO
   67 IF(NCV1(I).EQ.NTOC(II))GO TO 69
      WRITE(6,335)NCV1(I)
      WRITE(6,381)KOFO,(NTOC(II),II=1,KOFO)
      FLAG6=0
      GO TO 63
   69 CONTINUE
C*****************************************************************
C
C       IF ALL OF THE SAME CONSTRAINTS ARE RE-ACQUIRED
C       THEN ASSUME A CONSTRAINED EXTREMA AT XO
C
C*****************************************************************
   73 CONTINUE
      IF(KF+KO.NE.KOFO)GO TO 63
        CALL FUNCT(XT,YO,N1,NUM)
        CALL GRAD(XT,GRO,N1)
      FLAG3=1
      RETURN
   63 CONTINUE
C
C***** ONLY THOSE CONSTRAINTS WITH ALPHA'S EQUAL TO THE
C       LOWEST ALPHA WILL BE ADDED TO THE MANIFOLD.
C
      WRITE(6,304)ALPO,K,(NCV(NCS(I)),I=1,K)
  304 FORMAT('THE SMALLEST ALPHA=',E13.6,
     1' FOR',I3,' CONSTRAINTS.'/'THEY ARE',20I3)
C
```

```
C***** THESE XO'S ARE IN THE CONSTRAINT PLANE.
C
      DO 8 I=1,N1
      IF(FLAG4.EQ.0)GO TO 25
      XO(I)=XO(I)+ABS(ALPHA(NM))*P(I)
      GO TO 8
   25 CONTINUE
      XO(I)=XO(I)+ABS(ALPHA(NM))*HGRO(I)
    8 CONTINUE
      WRITE(6,410)N1,(XO(I),I=1,N1)
  410 FORMAT('THE NEW XO CALC. WITH ALPA IS'/NE13.6)
      CALL GRAD(XO,GRO,N1,NUM)
      CALL FUNCT(XO,YO,N1)
      WRITE(6,303)YO,N1,(GRO(I),I=1,N1)
  303 FORMAT('F(XO)=',E13.6/'GRAD F(XO)=',NE13.6//)
   18 CONTINUE
      KA=0
      KS=0
      KSS=0
      IF(KOF.NE.0)GO TO 47
      KOF=1
      MNB(0)=0
      IF(KF.GE.N1)GO TO 9
      GO TO 49
   47 CONTINUE
C
C***** SCAN THE DIFFERENCE BETWEEN THE OLD VIOLATED CONSTRAINTS
C       AND THOSE THAT ARE STILL BEING VIOLATED
C
      DO 27 I=1,KOF
      DO 26 J=1,KO
   26 IF(NCVO(I).EQ.NCV1(J))GO TO 27
      KS=KS+1
C
C***** KSS COUNTS ONLY THE NO. OF OLD CONSTRAINTS TO BE
```

```
C       DROPPED THAT ARE DETECTED BY THE CONSTRAINT TEST
C
        KSS=KSS+1
        MNB(KS)=I
   27 CONTINUE
C
C***** EXCESS CONSTRAINTS WILL BE DETECTED AND DROPPED
C
C*************************************************************
C*************************************************************
C       HERE WE ARE CONCERNED THAT SPACE IS COMPLETELY
C       SPANNED AND THE PROJECTION VECTOR IS TOTALLY DEFINED
C       LEAVING NO ORTHOGONAL COMPONENT. DROP WILL DETERMINE
C       THOSE CONSTRAINTS TO BE DROPPED.
C*************************************************************
C*************************************************************
        IF((KF+KO-KS).LT.N1)GO     TO 55
    9 CONTINUE
        CALL DROP(KO,NCV1,KF,NCV,NCS,FLAG3,FLAG6,P,XT,NTOC,KOFO,NUM)
        WRITE(6,381)KOFO,(NTOC(I),I=1,KOFO)
  381 FORMAT('IF',NI3,' ARE REAQUIRED AN EXTREMA IS ASSUMED')
        IF(FLAG3.EQ.1)GO     TO 65
        IF(FLAG6.EQ.3)GO TO 21
C*************************************************************
C                                                            *
C       IF FLAG6=3 THEN THE SEARCH VECTOR POINTS INTO FEASIBLE  *
C       SPACE. IF THERE ARE EQUALITY CONSTRAINTS PRESENT THEN   *
C       THEY ARE KEPT AND PROJECTIONS ARE MADE ON THEM. IF      *
C       NONE ARE PRESENT THEN RETURN TO THE DFP METHOD IS MADE.  *
C                                                            *
C*************************************************************
        RETURN
   55 CONTINUE
        IF(NEC.EQ.0.OR.KS.EQ.0)GO TO 71
        L=0
```

```
      KIND1=1
C
C***** WE MUST SCAN THE LIST OF CONSTRAINTS TO BE DROPPED TO DETERMINE
C       IF ANY OF THEM ARE EQUALITY CONSTRAINTS. IF ONE IS THEN
C       IT WILL DROPPED FROM THE LIST.
C
   54 CONTINUE
      DO 70 I=KIND1,KS
      DO 70 J=1,NEC
      IF(NCVO(MNB(I+L)).NE.(NLEC+J))GO TO 70
      IF(I.EQ.KS)GO TO 59
      L=L+1
      DO 61 KIND=I,KS-1
   61 MNB(KIND)=MNB(KIND+1)
      GO TO 58
   70 CONTINUE
      GO TO 71
   58 KS=KS-1
      KIND1=I
      GO TO 54
   59 KS=KS-1
   71 CONTINUE
      IF(KS.EQ.0)GO TO 49
      IF(KO.GT.0)GO TO 72
      WRITE(6,321)KS,(NCV(NCS(I)),I=1,KS)
      GO TO 49
   72 CONTINUE
      WRITE(6,321)KS,(NCVO(MNB(I)),I=1,KS)
  321 FORMAT(I3,' CONSTRAINTS WILL BE DROPPED. THEY ARE ',20I4)
   49 CONTINUE
      IF(KO.EQ.0)GO TO 29
C*************************************************************
C       AT THIS POINT WE UPDATE THE NCVO ARRAY BY
C       DROPPING OFF THE CONSTRAINT NOT STILL
C       VIOLATED. SINCE WE ASSUME ALL PLANES
```

```
C       TO BE LINEARLY INDEPENDENT THEN THE NUMBER OF
C       CONSTRAINTS VIOLATED CAN NEVER BE GREATER THAN
C       THE DIMENSIONS OF SPACE.
C*****************************************************************
      IDD=0
      DO 28 I=1,KO
      IF(KS.EQ.0)GO TO 57
      IF(KSS.EQ.0)KSS=1
      DO 56 J=KSS,KS
   56 IF(NCV1(I).EQ.NCVO(MNB(J)))GO TO 28
   57 CONTINUE
      IDD=IDD+1
      NCVO(IDD)=NCV1(I)
   28 CONTINUE
      KO=KO-KS
   29 CONTINUE
      IF(KS.EQ.0)GO TO 48
      NSN=-1
C
C***** THIS ROUTINE CALCULATES AND UPDATES THE PROJECTION MATRIX(AAT2)
C       THE ARGUMENT MNB HAS NO EFFECT IN SPAN FOR SUBTRACTION
C
      CALL SPAN(NCVO,MNB,IDD,NSN,IO)
   48 CONTINUE
      IF(KF.EQ.0)GO TO 39
      DO 44 I=1,KF
   44 NCVO(KO+I)=NCV(NCS(I))
      WRITE(6,322)KF,(NCV(NCS(I)),I=1,KF)
  322 FORMAT(I3,' CONSTRAINTS WILL BE ADDED. THEY ARE',20I4)
      NSN=1
      CALL SPAN(NCV,NCS,KF,NSN,IO)
   39 CONTINUE
      KC=KC+1
      DO 31 I=1,N1
   31 P(I)=0.0
```

```
      WRITE(6,333)
  333 FORMAT('THIS IS THE PRESENT PROJECTION MATRIX')
      WRITE(6,314)(N1,(AAT2(I,J),J=1,N1),I=1,N1)
      DO 40 J=1,N1
      DO 40 I=1,N1
C
C***** P IS THE PROJECTION OF THE GRAD ON THE CONSTRAINT
C     MANIFOLD. IT IS THE NEW SEARCH VECTOR
C
      IF(I.NE.J)GO TO 16
      P(I)=(1.-AAT2(I,J))*GRO(J)+P(I)
      GO TO 40
   16 P(I)=-AAT2(I,J)*GRO(J)+P(I)
   40 CONTINUE
      PT=0.0
      DO 42 I=1,N1
   42 PT=PT+P(I)**2
      KOF=KF+KO
      PT=SQRT(PT)
C
C***** IF PT=0 AND GRAD DOES NOT POINT INTO FEASIBLE SPACE
C      THEN A CONSTRAINED MAX(MIN) HAS BEEN MET.
C
      IF(PT.GT..001)GO TO 19
      FLAG3=1
      GO TO 65
   19 CONTINUE
C*************************************************************
C                                                            *
C    IF WE HAVE DONE A SMALL STEP INTO FEASIBLE SPACE        *
C    THEN WE WILL CHECK AT THIS POINT TO DETERMINE IF        *
C    THE NEW CONSTRAINT MANIFOLLD CONTAINS THE               *
C    EQUALITY CONSTRAINTS. THIS NEED ONLY BE DONE AT         *
C    A POINT WHERE THE PROJECTION VECTOR POINTS              *
C    DIRECTLY AWAY FROM THE ORIGINAL POINT WHERE ALL         *
```

```
C     OF THE CONSTRAINT PLANES INTERSECTED. THIS IS                    *
C     DETERMINED BY:                                                   *
C        +1=(XO-XT)/ABS(XO-XT)*P/PT                                    *
C                                                                      *
C**********************************************************************
      IF(FLAG6.EQ.0)GO TO 11
      XTD=0.
      XTDT=0.
      DO 12 I=1,N1
   12 XTD=XTD+(XO(I)-XT(I))**2
      XTD=SQRT(XTD)
      DO 30 I=1,N1
   30 XTDT=(XO(I)-XT(I))*P(I)/(XTD*PT)+XTDT
C
C***** CHECK TO SEE IF ALL CONSTRAINTS ARE GENERATED
C
      IF(XTDT.LT.0.0)GO TO 11
      IF(ABS(ABS(XTDT)-1.).GT.1.E-12)GO TO 11
      WRITE(6,387)ABS(ABS(XTDT)-1.)
  387 FORMAT('MUST BE > THAN 1.E-12 BUT IS',E14.6)
C
C***** SCAN MANIFOLD TO SEE IF ALL EQUALITY CONSTRAINTS ARE PRESENT
C
      M=0
      IF(KF.EQ.0)GO TO 33
      DO 32 I=1,KF
      IF(NCV(NCS(I)).LE.NLEC.OR.NCV(NCS(I)).GT.(NLEC+NEC))GO TO 32
      M=M+1
   32 CONTINUE
   33 IF(KO.EQ.0)GO TO 37
      DO 34 I=1,KO
      IF(NCV1(I).LE.NLEC.OR.NCV1(I).GT.(NLEC+NEC))GO TO 34
      M=M+1
   34 CONTINUE
   37 CONTINUE
```

```
C***** ALL EQUALITY CONSTRAINTS ARE PRESENT WHEN M=NEC
C
      IF(M.EQ.NEC)GO TO 36
      DO 35 I=1,N1
   35 X1(I)=XT(I)
      FLAG3=1
      RETURN
   36 FLAG6=0
      KOFO=0
   11 CONTINUE
      RETURN
   13 IF(FLAG2.EQ.1)GO     TO 17
      WRITE(6,302)
  302 FORMAT('NO CONSTRAINTS HAVE BEEN VIOLATED')
   10 FLAG2=0
      RETURN
   17 CONTINUE
C*****************************************************************
C       KO=0 THE CASE WHERE THE GRAD POINTS INTO FEAS. SPACE    *
C                                                               *
C       KO=KOF NO CONSTRAINTS WILL BE DROPPED FROM AAT2         *
C                                                               *
C       KO<KOF SOME CONSTRAINTS ARE TO BE DROPPED               *
C                                                               *
C       FLAG7=1 AN EXTREMA WAS FOUND ON THE LINE SEARCH         *
C*****************************************************************
      IF(KO.EQ.0)GO TO 21
      CALL GRAD(X1,GRT,N1,NUM)
      IF(FLAG7.EQ.1)GO TO 74
      IF(KO.LT.KOF)GO TO 18
      DO 79 I=1,N1
   79 XO(I)=X1(I)
      RETURN
   74 DO 78 I=1,N1
   78 GRO(I)=GRT(I)
```

```
      IF(KO.EQ.0)GO TO 29
      DO 27 I=1,KO
      X1(I)=0.
      AMT(I)=AMV
      IF(NCV1(I).LE.NLEC+NEC)AMT(I)=-AMV
      DO 24 J=1,N1
   24 X1(I)=X1(I)+A(NCV1(I),J)**2
   27 X1(I)=SQRT(X1(I))
      DO 25 I=1,KO
      DO 25 J=1,N1
      XO(J)=XO(J)+A(NCV1(I),J)/(X1(I)*AMT(I)*5.E5)
   25 CONTINUE
   29 CONTINUE
      IF(KF.EQ.0)GO TO 48
      DO 33 I=1,KF
      X1(I)=0.
      AMT(I)=AMV
      IF(NCV(NCS(I)).LE.NLEC+NEC)AMT(I)=-AMV
      DO 32 J=1,N1
   32 X1(I)=X1(I)+A(NCV(NCS(I)),J)**2
   33 X1(I)=SQRT(X1(I))
      DO 34 I=1,KF
      DO 34 J=1,N1
      XO(J)=XO(J)+A(NCV(NCS(I)),J)/(X1(I)*AMT(I)*5.E5)
   34 CONTINUE
   48 CONTINUE
      DO 35 I=1,N1
      NCVO(I)=0.0
   35 NCV1(I)=0.0
      WRITE(6,301)N1,(XO(I),I=1,N1)
  301 FORMAT('WE WILL START AT THE FOLLOWING POINT & FOLLOW THE GRAD'
     */NE13.6)
      RETURN
      END
```

```
      IF(KO.LT.KOF)GO TO 18
      DO 23 I=1,N1
   23 XO(I)=X1(I)
      GO TO 39
   21 IF(NEC.NE.0)GO TO 75
      FLAG2=2
      KOF=0
   65 CONTINUE
      DO 53 I=1,N1
      X1(I)=XO(I)
      DO 53 J=1,N1
   53 AAT2(I,J)=0.0
      RETURN
C*****************************************************************
C       IF THE GRAD POINTS INTO FEASIBLE SPACE IT WILL          *
C       STILL HAVE TO SATISFY THE EQUALITY CONSTRAINTS          *
C       HENCE THE GRAD WILL CONTINUE TO BE PROJECTED            *
C       ONTO THE EQUALITY CONSTRAINTS. ONLY THE EQUALITY        *
C       CONSTRAINTS MANIFOLD WILL BE KEPT.                      *
C*****************************************************************
   75 DO 76 I=1,N1
      DO 76 J=1,N1
   76 AAT2(I,J)=0.0
      IO=0
      KO=0
      KF=NEC
      DO 77 I=1,KF
      NCV(I)=NLEC+I
      NCS(I)=I
      NCVO(I)=NLEC+I
   77 NCV1(I)=NLEC+I
      GO TO 48
      END
```

```
      SUBROUTINE SPAN(NO,NOC,KN,NSN,TO)
C*****************************************************************
C
C       THIS ROUTINE CALCULATES AND UPDATES THE PROJECTION MATRIX(AAT2)
C       IT USES THE GRAM-SCHMIDT METHOD AND IS WRITTEN IN THE
C       USUAL MATRIX NOTATION.
C
C*****************************************************************
      COMMON X0(10),X1(10),H(10,10),A(20,10),GRO(10),BC(20),
     1N1,NC,AAT2(10,10),KOF,NCVO(20),NLEC,NEC,NGEC,NGEX,KC
      DIMENSION AL(10,10),NOC(10),NO(10),IDPAST(10)
      REAL IDENT
C
C***** IF NSN<0 SET AAT2=0.0 THEN RECALCULATE AAT2 WITHOUT
C       THE DROPPED CONSTRAINTS.
      IF(IO.EQ.0)IT=0
      IO=1
      JK=1
      IF(NSN.LT.0)GO TO 10
   12 CONTINUE
      DO 1 J=JK,KN
      DO 1 K=1,N1
    1 AL(J,K)=0.0
    9 CONTINUE
      DO 7 I=JK,KN
      IF(NSN.GT.0)GO TO 14
      NOC(I)=I
   14 CONTINUE
      II=II+1
      IDPAST(II)=NO(NOC(I))
      WRITE(6,304)N1,(A(NO(NOC(I)),J),J=1,N1)
  304 FORMAT('THIS IS THE PRESENT CONSTRAINT EQ. IN SPAN'/NE13.7)
      DO 3 J=1,N1
      DO 3 K=1,N1
      IF(J.NE.K)GO TO 23
```

```
      AL(I,J)=(1.0-AAT2(J,K))*A(NO(NOC(I)),K)+AL(I,J)
      GO TO 3
   23 AL(I,J)=-AAT2(J,K)*A(NO(NOC(I)),K)+AL(I,J)
    3 CONTINUE
      ALABS=0.0
      DO 4 K=1,N1
    4 ALABS=ALABS+AL(I,K)**2
      ALABS=SQRT(ALABS)
      IF(ALABS.LT..0000000001)GO TO 7
      DO 5 J=1,N1
    5 AL(I,J)=AL(I,J)/ALABS
      DO 6 K=1,N1
      DO 6 L=1,N1
      IF(L.LT.K)GO TO 19
      AAT2(K,L)=AAT2(K,L)+AL(I,K)*AL(I,L)
   19 AAT2(L,K)=AAT2(K,L)
    6 CONTINUE
    7 CONTINUE
      RETURN
C**** SUBTRACT DIMENSIONS TO LOWEST ONE TO BE REMOVED
   10 CONTINUE
      DO 11 J=1,N1
      DO 11 K=1,N1
   11 AAT2(J,K)=0.0
      DO 2 J=1,KN
    2 IF(NO(NOC(J)).LT.(II-KN))GO TO 16
      DO 17 I=II,II-KN,-1
      DO 17 K=1,N1
      DO 17 L=1,N1
      IF(L.LT.K)GO TO 20
      AAT2(K,L)=AAT2(K,L)-AL(I,K)*AL(I,L)
   20 AAT2(L,K)=AAT2(K,L)
   17 CONTINUE
      RETURN
   16 CONTINUE
```

```
      JK=II
      DO 8 J=1,II
      DO 8 K=1,KN
      IF(NO(NOC(K)).NE.IDPAST(J))GO TO 8
      IF(J.GE.JK)GO TO 8
      JK=J
    8 CONTINUE
C
C     JK IS EQUAL TO THE LOWEST AL TO BE DROPPED. THE ONES BELOW THIS
C     NEED NOT BE RECALCULATED.
C
      IF(JK.LT.2)GO TO 15
      DO 13 I=1,JK-1
      DO 13 K=1,N1
      DO 13 L=1,N1
      IF(L.LT.K)GO TO 21
      AAT2(K,L)=AAT2(K,L)+AL(I,K)*AL(I,L)
   21 AAT2(L,K)=AAT2(K,L)
   13 CONTINUE
   15 II=JK-1
      GO TO 12
      END
```

```
      SUBROUTINE DROP(KO,NCV1,KF,NCV,NCS,FLAG3,FLAG6,P,XT,NTOC,KOFO,NUM)
      COMMON XO(10),X1(10),H(10,10),A(20,10),GRO(10),BC(20),
     1N1,NC,AAT2(10,10),KOF,NCVO(20),NLEC,NEC,NGEC,NGEX,KC
      DIMENSTON NCV1(10),NCV(10),NCS(10),P(10),XT(10),AMT(10),
     *UGRO(10),NTOC(10)
      REAL LAMDA
      INTEGER FLAG3,FLAG6,FLAG7
C*************************************************************
C
C       FLAG6=1 ALERTS CHK THAT A NEW POINT IS BEING RETURNED
C               AND IT IS FEASIBLE SPACE.
C
C             =2 ALERTS CHK THAT THE NEW POINT IS IN
C                INFEASIBLE SPACE.
C
C             =3 ALERTS CHK THAT THE NEW POINT IS IN INFEASIBLE SPACE
C                AND ONLY THE EQUALITY CONSTRAINTS ARE KEPT
C
C       FLAG7=1 WHEN THERE UNIT CONSTRAINTS PRESENT
C
C*************************************************************
      FLAG7=0
      KOFO=KF+KO
      B=0
      DO 28 I=1,N1
      P(I)=GRO(I)
   28 B=B+GRO(I)**2
      AGRO=SQRT(B)
      DO 43 I=1,N1
   43 UGRO(I)=GRO(I)/AGRO
      K=0
      NT=NLEC+NEC+NGEC
      IF(KO.EQ.0)GO TO 3
      DO 2 I=1,KO
      IF(NCV1(I).GT.NLEC.AND.NCV1(I).LE.NLEC+NEC)FLAG7=1
```

```
      NTOC(I)=NCV1(I)
    2 CONTINUE
    3 CONTINUE
      IF(KF.EQ.0)GO TO 45
      DO 5 I=1,KF
      IF(NCV(NCS(I)).GT.NLEC.AND.NCV(NCS(I)).LE.NLEC+NEC)FLAG7=1
      NTOC(I+KO)=NCV(NCS(I))
    5 CONTINUE
   45 CONTINUE
C
C***** AMT IS USED TO SET THE SENSE OF THE NORMALS
C       AMT>0 WHEN CONSTRAINT IS LE TYPE & GRAD POINTS TO
C                   FEASIBLE SPACE
   21 FLAG6=1
      WRITE(6,330)NTOC(I),I=1,N1
  330 FORMAT('THE CONSTRAINTS TO BE DROPPED ARE',20I3)
      AMV=1.
      IF(KO.EQ.0)GO TO 42
      DO 37 I=1,KO
      IF(NCV1(I).GT.NLEC.AND.NCV1(I).LE.(NLEC+NEC))GO TO 37
      B=0.
      DO 36 J=1,N1
   36 B=B+A(NCV1(I),J)*(XO(J)+UGRO(J)*.5E-5)
C 314 FORMAT('I,NCV1(I),BC,B',2I3,2E13.6)
      IF(BC(NCV1(I))-B.LT.0.AND.NCV1(I).LE.NLEC)GO TO 41
      IF(BC(NCV1(I))-B.GT.0.AND.NCV1(I).GT.NLEC+NEC)GO TO 41
   37 CONTINUE
   42 CONTINUE
      IF(KF.EQ.0)GO TO 47
      DO 39 I=1,KF
      IF(NCV(NCS(I)).GT.NLEC.AND.NCV(NCS(I)).LE.NLEC+NEC)GO TO 39
      B=0.
      DO 38 J=1,N1
   38 B=B+A(NCV(NCS(I)),J)*(XO(J)+UGRO(J)*.5E-5)
      IF(BC(NCV(NCS(I)))-B.LT.0.AND.NCV(NCS(I)).LE.NLEC)GO TO 41
```

```
      IF(BC(NCV(NCS(I)))-B.GT.0.AND.NCV(NCS(I)).GT.NLEC+NEC)GO  TO 41
   39 CONTINUE
   47 CONTINUE
   40 AMV=-1.
      FLAG6=2
   41 CONTINUE
C*****************************************************************
C*****************************************************************
C
C       FROM 21 TO HERE IS USED TO SORT THE CONDITION
C       WHERE THE GRAD POINTS INTO FEAS. SPACE FROM X0
C       AMV=-1 WHEN GRAD POINTS INTO FEAS. SPACE.  THIS IS
C       KNOWN WHEN THE ONLY CONSTRAINT VIOLATED IS AN EQUALITY
C
C*****************************************************************
C*****************************************************************
      DO 26 I=1,N1
      P(I)=GR0(I)
      XI(I)=X0(I)
      DO 26 J=1,N1
   26 AAT2(I,J)=0.0
      KOF=0
      IF(FLAG7.NE.1)GO TO 44
      WRITE(6,318)
  318 FORMAT('ONLY THE EQUALITY CONSTRAINTS WILL BE KEPT')
      FLAG6=3
      RETURN
C*****************************************************************
C
C       THE REST OF THE SUBROUTINE CALC'S A VECTOR THAT IS THE
C       AVERAGE OF ALL THE CONSTRAINT NORMALS AND IS GIVEN A
C       DIRECTION OPPOSITE TO THE GRADIENT VECTOR.
C
C*****************************************************************
   44 CONTINUE
```

```
      SUBROUTINE FEASPT(FLAG5)
C*****************************************************************
C       THIS ROUTINE BUILDS A SIMPLEX TABLEAU CALLED THE AUG MATRIX
C       THE MATRIX IS CONSTRUCTED ROW BY ROW AND IS STRUCTURED AS FOLLOWS:
C            AUG = [ A : B : C ]
C       A - THIS IS THE CONSTRAINT MATRIX WITHOUT X'S>0
C       B - THIS IS A DIAG. MATRIX THAT CONSISTS OF THE POS.
C           SLACK'S(NLEC) AND THE POS. ARTIFICIALS(NEC).
C       C - THIS IS A MATRIX OF ALL THE NEG. SLACKS(NGEC).
C*****************************************************************
      COMMON XO(10),X1(10),H(10,10),A(20,10),GRO(10),BC(20),
     1N1,NC,AAT2(10,10),KOF,NCVO(20),NLEC,NEC,NGEC,NGEX,KC
      DIMENSION C1(30),C2(30),AUG(10,30),BCC(20),IPVTK(20),CB(20)
      WRITE(6,310)NC,NGEX,NLEC,NEC,N1
  310 FORMAT('NC,NGEX,NLEX,NEC,N1',5I3)
      INTEGER FLAG5,IPVTK
C
C***** IF THE NO. OF EQUALITY CONSTRAINTS IS EQUAL TO THE PROBLEM
C       DIMENSIONS.
C
      IF(NEC.LT.N1)GO TO 27
   22 WRITE(6,308)
  308 FORMAT('TOO MANY EQUALITY CONSTRAINTS FOR THE DEGREES OF FREEDOM')
      FLAG5=1
      RETURN
   27 CONTINUE
      NC1=NC-NGEX
      DO 4 I=1,NC1
      BCC(I)=BC(I)
      IPVTK(I)=0
C
C     LOAD A INTO AUG(A) ROW I
C
      DO 1 J=1,N1
    1 AUG(I,J)=A(I,J)
```

```
C
C      LOAD 1 INTO AUG(B) IN THE PROPER DIAGONAL POSITION
C
       DO 2 J=1,NC1
       AUG(I,N1+J)=0.0
     2 IF(I.EQ.J)AUG(I,N1+J)=1.0
       IF(NGEC.EQ.0)GO TO 4
C
C      LOAD -1 INTO AUG(C) FOR THE ARTIFICIALS OF GE CONSTRAINTS
C
       DO 3 J=1,NGEC
       AUG(I,NC1+N1+J)=0.0
     3 IF(I.GT.(NLEC+NEC).AND.(I-(NLEC+NEC)).EQ.J)AUG(I,N1+NC1+J)=-1.0
     4 CONTINUE
       DO 7 I=1,NC1+N1+NGEC
       C1(I)=0.0
C
C       THIS ALLOWS THE "CJ" ROW TO CONTAIN ONLY 0 EXCEPT
C       WHERE THERE WAS AN = TYPE CONSTRAINT OR WHERE
C       AN ARTIFICIAL HAS BEEN ADDED. THE "CJ" ROW IS
C       TERMED C1 IN THE PROGRAM. C2 IS THE COST ROW.
C
       IF(I.LE.(N1+NLEC).OR.I.GT.(N1+NC1))GO TO 7
       C1(I)=1.0
     7 CONTINUE
       DO 30 I=1,NC1
    30 CB(I)=C1(I+N1)
    17 CONTINUE
       WRITE(6,301)
   301 FORMAT('AUG MATRIX')
       WRITE(6,302)(N1,(AUG(II,JJ),JJ=1,N1),II=1,NC1)
   302 FORMAT(NE10.3)
       WRITE(6,302)(NC1+NGEC,(AUG(II,JJ),JJ=1+N1,NC1+NGEC+N1),II=1,NC1)
C
C***** CALCULATE THE COST ROW
```

```
C
      DO 8 I=1,NC1+N1+NGEC
      C2(I)=0.0
C
C       J IS THE ELEMENT NO. IN COL. I
C
      DO 5 J=1,NC1
    5 C2(I)=C2(I)+CB(J)*AUG(J,I)
      C2(I)=C1(I)-C2(I)
    8 CONTINUE
      WRITE(6,330)
  330 FORMAT('THIS IS THE COST ROW')
      WRITE(6,304)C2(LL),LL=1,NC1+N1+NGEC
  304 FORMAT(10E10.3)
C
C***** THE COST ROW IS SCANNED TO DETERMINE IF IN FEASIBLE SPACE
C
      DO 9 I=1,NC1+N1+NGEC
      IF(C2(I).GE.0)GO TO 9
C
C***** IF A NEG COST IS FOUND THEN THE PT. NOT YET FEASIBLE
C
      GO TO 10
    9 CONTINUE
C
C***** THIS IS A FEASIBLE PT. CANDIDATE
C
      GO TO 18
   10 CONTINUE
      I=1
      DO 11 J=1,NC1+N1+NGEC
C
C       THE COST ROW IS SCANNED FOR THE LOWEST VALUE
C       THE I'TH COL. IS THE PIVOT COL.
C
```

```
   11 IF(C2(J).LT.C2(I))I=J
C
C       CALCULATE THE RATIO'S AND FIND THE PIVOT ELEMENT
C       BY FINDING THE ROW WITH THE LOWEST RATIO
C
        BC12=1.E19
        J=1
        DO 12 L=1,NC1
        IF(AUG(L,I).GT.0.)GO TO 24
        BC1=1.E20
        GO TO 23
   24   BC1=BCC(L)/AUG(L,I)
   23   CONTINUE
        IF(BC1.GE.BC12)GO TO 12
        J=L
        BC12=BC1
   12   CONTINUE
        IF(BC12.LE.1.E18)GO TO 28
        WRITE(6,315)I
  315   FORMAT('COLUMN',I3,' IS ALL NEGATIVE OR ZERO')
        FLAG5=1
        RETURN
   28   CONTINUE
C
C***** IPVTK KEEPS TRACK OF THE ELEMENTS IN THE BASIS. IPVTK(J)=I
C       MEANS THE J'TH BASIS VARIABLE =THE I'TH VARIABLE
C
        IPVTK(J)=I
C
C***** AUG(J,I) IS THE PIVOT ELEMENT AND THE J'TH BASE VAR.
C       IS REPLACED BY THE I'TH VARIABLE
C***** NOW NORMALIZE THE PIVOT ROW
C
        BCC(J)=BCC(J)/AUG(J,I)
        AUGT=AUG(J,I)
```

```
      DO 14 L=1,NC1+N1+NGEC
   14 AUG(J,L)=AUG(J,L)/AUGT
C
C***** THE NEXT TABLEAU WILL NOW BE DEFINED
C
      DO 16 L=1,NC1
C
C***** SKIP THE PIVOT ROW
C
      IF(L.EQ.J)GO TO 16
      AUGT=AUG(L,I)
      BCC(L)=BCC(L)-BCC(J)*AUGT
      WRITE(6,314)L,J,BCC(L),BCC(J),AUGT
      DO 15 K=1,NC1+N1+NGEC
      AUG(L,K)=AUG(L,K)-AUG(J,K)*AUGT
  314 FORMAT('BCC(L)=BCC(L)-BCC(J)*AUGT'/2T3,3E13.6)
   15 CONTINUE
   16 CONTINUE
      WRITE(6,305)J,I
  305 FORMAT('PIVOT ELEMENT IS',2T3)
C
C     AT THIS POINT THE BASIS VARIABLES MULTIPLIER COEFFICENTS
C     ARE UPDATED.
C
      CB(J)=C1(I)
      GO TO 17
   18 CONTINUE
C
C***** THIS IS A FEASIBLE PT. CANDIDATE
C***** ALL BCC'S WILL BE SCANNED TO BE SURE THEY ARE POSITIVE
C
      DO 19 I=1,NC1
      IF(BCC(I).GE.0.0)GO TO 19
      GO TO 21
   19 CONTINUE
```

```
C
C***** SET THE INITIAL FEASIBLE POINT
C
      DO 20 I=1,NC1
   20 XO(I)=0.0
      DO 29 I=1,NC1
      IF(IPVTK(I).GT.O.AND.IPVTK(I).LE.N1)XO(IPVTK(I))=BCC(I)
   29 CONTINUE
      WRITE(6,320)
  320 FORMAT('THIS THE INITIAL POINT')
      WRITE(6,321)N1,(XO(I),I=1,N1)
  321 FORMAT(NE13.6)
      RETURN
   21 WRITE(6,300)
  300 FORMAT('NO INITIAL FEASIBLE POINT FOUND')
      FLAG5=1
      RETURN
      END
```

1. REPORT NO.	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.
NASA TP 2086		

4. TITLE AND SUBTITLE	5. REPORT DATE
Nonlinear Optimization with Linear Constraints Using a Projection Method	September 1982
	6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S)	8. PERFORMING ORGANIZATION REPORT #
Thomas Fox	M-389
9. PERFORMING ORGANIZATION NAME AND ADDRESS	10. WORK UNIT NO.
George C. Marshall Space Flight Center Marshall Space Flight Center, Alabama 35812	11. CONTRACT OR GRANT NO.
	13. TYPE OF REPORT & PERIOD COVERED
12. SPONSORING AGENCY NAME AND ADDRESS	Technical Paper
National Aeronautics and Space Administration Washington, D.C. 20546	14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

Prepared by Systems Dynamics Laboratory, Science and Engineering Directorate.

16. ABSTRACT

This report examines and discusses nonlinear optimization problems that are encountered in science and industry. A new method of projecting the gradient vector onto a set of linear constraints is developed, and a program that uses this method is presented. The algorithm that generates this new projection matrix is based on the Gram-Schmidt method and overcomes some of the objections to the Rosen projection method. This should make the projection method of optimization with linear constraints more attractive to users.

17. KEY WORDS	18. DISTRIBUTION STATEMENT
Nonlinear optimization with linear constraints Projection method DFP search techniques	Unclassified - Unlimited STAR Category: 66

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
Unclassified	Unclassified	97	A05

For sale by National Technical Information Service, Springfield, Virginia 22161
NASA-Langley, 1982

National Aeronautics and Space Administration

Washington, D.C.
20546

Official Business
Penalty for Private Use, $300

THIRD-CLASS BULK RATE

Postage and Fees Paid
National Aeronautics and
Space Administration
NASA-451

3 1 1U,G, 820923 S00903DS
DEPT OF THE AIR FORCE
AF WEAPONS LABORATORY
ATTN: TECHNICAL LIBRARY (SUL)
KIRTLAND AFB NM 87117

S

NASA

POSTMASTER: If Undeliverable (Section 158 Postal Manual) Do Not Return